Warsaw University
Faculty of Mathematics, Informatics and Mechanics

Ngo Chi Lang
Index: 181191

A tolerance rough set approach to clustering web search results

Master thesis
in COMPUTER SCIENCE

Supervisor
dr Nguyen Hung Son
Institute of Mathematics
Faculty of Mathematics, Informatics and Mechanics
Warsaw University

December 2003

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science

Date Author:

Ready for review

Date Supervisor:

Abstract

Searching for information on the Web has attracted great attention in many research communities. Due to the enormous size of the Web and low precision of user queries, results returned from present web search engines can reach hundreds or even hundreds of thousands documents. Therefore, finding the right information can be difficult if not impossible. One approach that tries to solve this problem is by using clustering techniques for grouping similar document together in order to facilitate presentation of results in more compact form and enable thematic browsing of the results set. In this thesis, a survey of recent document clustering techniques is presented with emphasis on application to web search results. An algorithm for web search results clustering based on Tolerance Rough Set is presented and its practical implementation is discussed. The proposed solution is evaluated in search results returned from actual web search engines and compared with other recent methods.

Keywords

clustering, grouping, search engine, search results, tolerance, rough set, algorithm

Classification

H. Information Systems
H.3 Information Storage and Retrieval
H.3.3 Information Search and Retrieval
Subjects: Clustering

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1. Motivation

With its explosive growth, the World Wide Web (the Web) has become an immense resource of textual data, images and other multimedia content. For efficient access and exploration of useful information, appropriate interfaces to search and navigation through this enormous collection are of critical need.

1.2. Searching the Web

Two most popular approaches to facilitate searching for information on the Web are presented by web search engine and web directories. Web search engines allow user to formulate a query, to which it responds using its index to return set of references to relevant web documents (web pages). Some of most popular search engines are Google[1], Altavista[2], HotBot[3], Lycos[4], AllTheWeb[5]. Web directories like Yahoo[6], Open Directory Project[7] are human-made collection of references to web documents organized as hierarchical structure of categories. User can search for information by navigating through categories to pinpoint the needed reference.

Although the performance of search engines is improving every day, searching on the Web can be a tedious and time-consuming task due to the following facts:

- search engines can only index part of the "indexable" Web, due to the huge size and highly dynamic nature of the Web.

- the user's "intention behind the search" is not clearly expressed which resulted in too general, short queries

- as the effects of the two above, results returned by search engine can count from hundreds to hundred thousands of documents

[1]http://www.google.com
[2]http://www.altavista.com
[3]http://www.hotbot.com
[4]http://www.lycos.com
[5]http://www.alltheweb.com
[6]http://www.yahoo.com
[7]http://www.dmoz.org

1.3. Search results clustering

One approach to manage large results set is by clustering. The concept arises from document clustering in Information Retrieval domain: *find a grouping for a set of documents so that documents belonging to the same cluster are similar and documents belonging to different cluster are dissimilar.* Search results clustering thus can be defined as *a process of automatical grouping search results into to thematic groups.* However, in contrast to traditional document clustering, clustering of search results are done on-the-fly (per user query request) and locally on a limited set of results return from the search engine.

Clustering of search results can help user navigate through large set of documents more efficiently. By providing concise, accurate description of clusters, it lets user localizes interesting document faster.

The earliest work on clustering results were done by Pedersen, Hearst et al. on Scather/Gather system [7, 11], followed with application to web documents and search results by Zamir et al. [43, 44] to create Grouper based on novel algorithm Suffix Tree Clustering. Inspired by their work, a Carrot framework was created by Weiss [38] to facilitate research on clustering search results. This has encouraged others to contribute new clustering algorithms under the Carrot framework like LINGO [19], AHC [40]. Other clustering algorithms were proposed by Zhang [46], Semantic Hierarchical Online Clustering using Latent Semantic Indexing to cluster Chinese search results or Class Hierarchy Construction Algorithm by Shenker et al. [1].

In this thesis, we proposed an approach to search results clustering based on Tolerance Rough Set following the work on document clustering of Bao [26, 34]. Tolerance classed are used to approximate concepts existed in documents. Set of documents sharing similar concepts are grouped together to form clusters. Concise, intelligible cluster labels are next derived from tolerance classes using special heuristic.

1.4. Layout of the thesis

This thesis structure is organized as follows. In chapter 2, the problem of searching for information on the web will be described in details with a special emphasis on issues related to results presentation and user interfaces of search engines. The chapter continues with a study of previous work on document clustering algorithms and its special application to clustering of search results.

The idea of clustering is presented in chapter 3, with emphasis on document clustering. Important aspects of document clustering algorithms are discussed. In chapter 4, preliminary concepts of rough set theory are introduced and a description of generalized tolerance space is presented. Following this, the Tolerance Rough Set model and its application to document clustering is presented.

In chapter 5, the Tolerance Rough Clustering algorithm for clustering search results is described in details and an approach to identify concise labels for clusters is proposed. Implementation details of the proposed algorithm within Carrot architecture is also demonstrated.

Evaluation results obtained from experiments with the algorithm is discussed in chapter 6. Finally, in chapter 7, conclusion drawn from the thesis and directions for future research will be given.

Chapter 2

Searching for information on the Web

The World Wide Web has become *de facto* the largest online library available practically to anyone with access to internet. For many of us, it is the one most valuable resource of information, without which every day activities would be unimaginably ineffective. Searching for information on the Web is a challenging task that has attracted great attention from many research communities.

2.1. Search engines

Web search engines (or web search services) exemplify the most popular way to access information in the Web. The usage of search engines is based on a simple paradigm: for a query asked by a user, search engine responds with a list of search results (also called snippets). *Snippet* is a short description of a web document (web page). It is usually composed from a title, a web address of a document (called URL[1]) and a short text highlighting content of referred web document (see Figure 2.2).

The general scheme of how a search engine works can be described as follows:

- **crawler** (or a set of it) starts from set of URLs, collects web pages represented by those URLs, storing them in a **document base**; contents of collected pages are next scanned for new URLs which will served as next points for crawlers to explore.

- an **indexer** create an indexing structure (inverted index, link-analysis structure, etc.) as **document index** from pages in **document base**

- for a user request, a **query processor** uses **document index** to find relevant documents and creates an ordered list of documents based on its ranking algorithm

- the ranked-list of documents is next returned and presented to user in the **user interface**

[1]URL – Uniform Resource Locator is a standard for specifying the location of a resource (e.g. file or web site) on the Internet (e.g. http://www.yahoo.com)

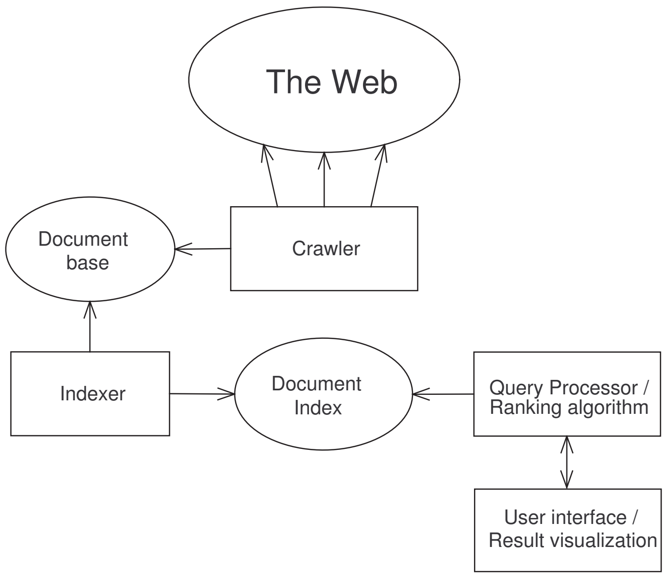

The Web
Crawler
Document base
Indexer
Document Index
Query Processor / Ranking algorithm
User interface / Result visualization

Figure 2.1: General architecture of a search engine

2.1.1. Ranking algorithms

Although all areas of search engine technology had been actively developed in the research community, it is ranking algorithm that has been in center of attention.

First generation search engines like Infoseek [12], Excite [9], Altavista [4] used algorithms that ranks documents primarily based on its textual contents. This approach has serious deficiencies as one can effectively abuse search engine ranking algorithm, pushing own authored web page to the top ranking by excessively populating web document with number of terms not necessarily related to document original contents (spamming). This has led to the invention of link-based ranking techniques, which had eliminated spamming almost entirely.

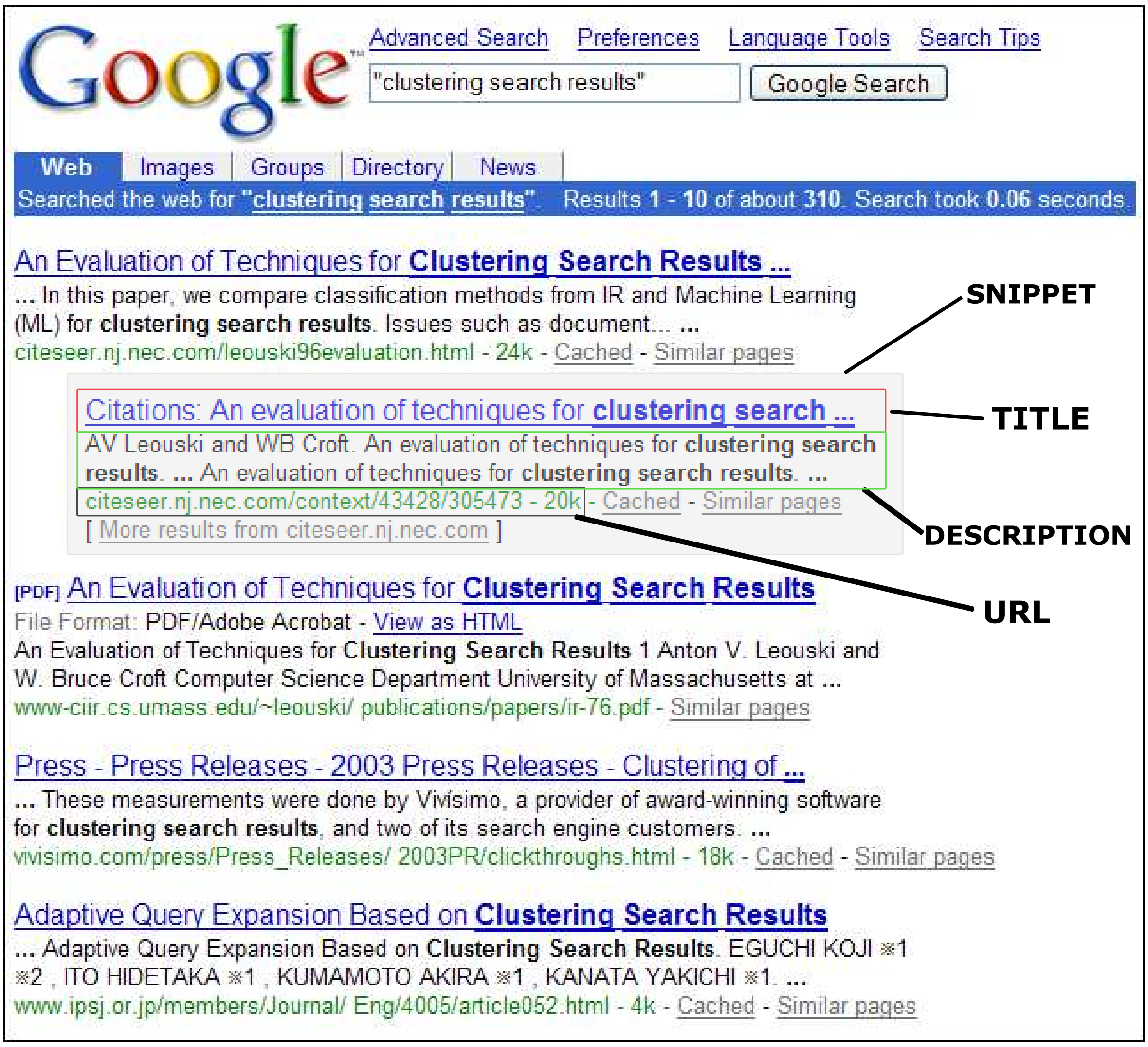
Second generation search engines use link analysis to rank web pages according to its linking popularity. The Web is modelled as a massive graph in which nodes are the pages, and directed edges symbolize links from one page to another. In this model, the more a page has incoming links, the higher it is ranked. Moreover, the page's ranking is taken into consideration when calculating ranking for pages that is has outgoing links to.

2.2. User interaction

Despite the great improvement in performance of search engines ranking algorithm, relatively less efforts has been put into improving *user search interface*. Exploring new search visualization paradigms is a promising direction in towards overall improvement of user's search experience.

2.2.1. Query - concept mismatch

The main issue in user interaction with search engines is the mismatch between what user wants to look for and what user formulates as a query to search engine (see Figure 2.3). When user wants to find some information on the Web, he/she has a concept of what he or she wants to search for. This concept is next translated by the user to a set of words that



Advanced Search Preferences Language Tools Search Tips

"clustering search results" Google Search

Web Images Groups Directory News

Searched the web for **"clustering search results"**. Results **1** - **10** of about **310**. Search took **0.06** seconds.

An Evaluation of Techniques for **Clustering Search Results ...**
... In this paper, we compare classification methods from IR and Machine Learning (ML) for **clustering search results**. Issues such as document... ...
citeseer.nj.nec.com/leouski96evaluation.html - 24k - Cached - Similar pages

Citations: An evaluation of techniques for **clustering search ...**
AV Leouski and WB Croft. An evaluation of techniques for **clustering search results**. ... An evaluation of techniques for **clustering search results**. ...
citeseer.nj.nec.com/context/43428/305473 - 20k - Cached - Similar pages
[More results from citeseer.nj.nec.com]

[PDF] An Evaluation of Techniques for **Clustering Search Results**
File Format: PDF/Adobe Acrobat - View as HTML
An Evaluation of Techniques for **Clustering Search Results** 1 Anton V. Leouski and W. Bruce Croft Computer Science Department University of Massachusetts at ...
www-ciir.cs.umass.edu/~leouski/ publications/papers/ir-76.pdf - Similar pages

Press - Press Releases - 2003 Press Releases - Clustering of ...
... These measurements were done by Vivisimo, a provider of award-winning software for **clustering search results**, and two of its search engine customers. ...
vivisimo.com/press/Press_Releases/ 2003PR/clickthroughs.html - 18k - Cached - Similar pages

Adaptive Query Expansion Based on **Clustering Search Results**
... Adaptive Query Expansion Based on **Clustering Search Results**. EGUCHI KOJI ※1 ※2 , ITO HIDETAKA ※1 , KUMAMOTO AKIRA ※1 , KANATA YAKICHI ※1. ...
www.ipsj.or.jp/members/Journal/ Eng/4005/article052.html - 4k - Cached - Similar pages

Figure 2.2: A ranked-list of search results from Google [10]

will be used as a query for the search engine. The preciseness of concept's translation can vary greatly, depending on many factors such as user experience in searching, background knowledge about the domain of concept or language proficiency.

Studies of search engine's log [27, 29] reveal that web search queries are on average only between 2 to 3 words long. For these imprecise queries, the number of results returned by web search engine can count from hundreds to hundreds of thousands document, thus finding the needed document amongst this set can be a difficult task.

2.2.2. Query expansion

An approach adapted by search engines to help user in formulating their queries is query expansion [5]. Its goal is to improve precision of user query by suggesting further keywords to be added to original one. It potentially leads to narrower search results and can direct user towards their "intended concept"[2]. Variations of query expansion techniques can select additional keywords locally [41, 42] — within the set of retrieved documents — or globally

[2]concept user initially wanted to search for but didn't know how to put it in query words

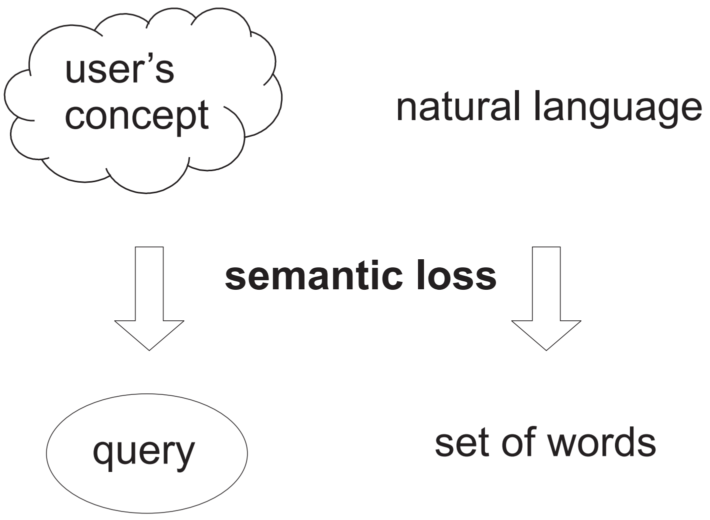


Figure 2.3: Semantic loss in transformation of concept into query

[41] — identifying terms closed to the words in query based on the whole document corpus. Other approaches adapt techniques for mining search engine's past usage log to find and use similar queries as a suggestion. Search engines that use query expansion to refine user 's search are Altavista [4], AllTheWeb [3].

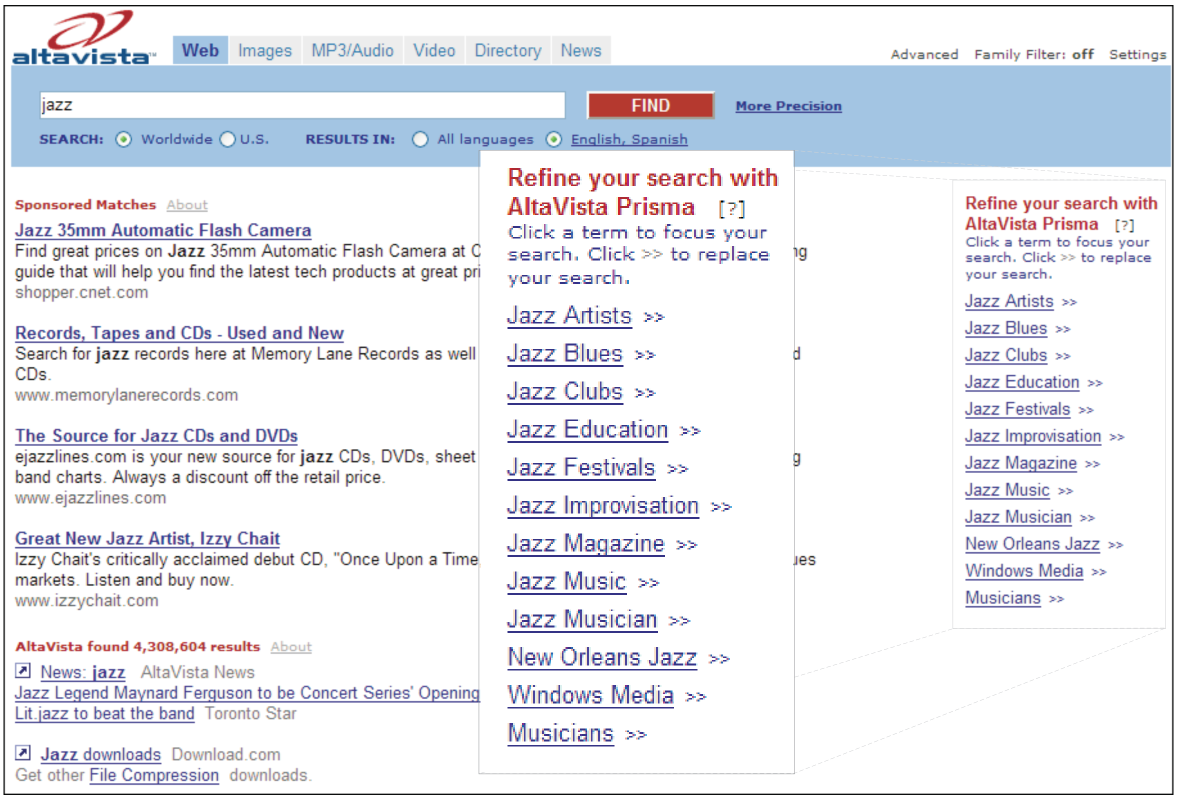
2.2.3. Ranked list

Most search engines present the results of user query in a form of a list of snippets, ranked by its relevancy to user query. When there are more than a couple of dozen results, ranked-list becomes impractical to browse and navigate. Analysis of search engine logs [15] shows that *"over half of users did not access result beyond the first page and more than three in four users did not go beyond viewing two pages"*. Along with the fact that by default search engines display from 10 to 20 results per page, this means that the large majority of users is willing to browse through at most 10 to 30 results.

Another issue with ranked-list is the force of strict ordering between search results, which is based on the assumption that snippets are always comparable using some measure. For general topic queries, this may not be a case as the results set tends to contain documents from sub-topics, between which comparison is inadequate (e.g. for query "jaguar", the result set may include items about jaguar cars, jaguar as a cat, Mac OS X Jaguar, etc.). In consequence, the results relevant to user needs can be positioned far further in the list.

Graphic-based interface

Several alternatives to text-based results presentation has been proposed, like Kartoo [17] that uses cartographic interface to show results as points-islands on the map. In Self-Organizing Maps (SOM), encoded representation of document is placed on two-dimensional map, where nearby locations contain similar documents. SOM has been used to visualize collection of web documents in WEBSOM [33], Map.net [32].



Sponsored Matches About

Jazz 35mm Automatic Flash Camera
Find great prices on **Jazz** 35mm Automatic Flash Camera at C
guide that will help you find the latest tech products at great pri
shopper.cnet.com

Records, Tapes and CDs - Used and New
Search for **jazz** records here at Memory Lane Records as well
CDs.
www.memorylanerecords.com

The Source for Jazz CDs and DVDs
ejazzlines.com is your new source for **jazz** CDs, DVDs, sheet
band charts. Always a discount off the retail price.
www.ejazzlines.com

Great New Jazz Artist, Izzy Chait
Izzy Chait's critically acclaimed debut CD, "Once Upon a Time
markets. Listen and buy now.
www.izzychait.com

AltaVista found 4,308,604 results About
News: jazz AltaVista News
Jazz Legend Maynard Ferguson to be Concert Series' Opening
Lit jazz to beat the band Toronto Star

Jazz downloads Download.com
Get other File Compression downloads.

Refine your search with AltaVista Prisma [?]
Click a term to focus your search. Click >> to replace your search.

Jazz Artists >>
Jazz Blues >>
Jazz Clubs >>
Jazz Education >>
Jazz Festivals >>
Jazz Improvisation >>
Jazz Magazine >>
Jazz Music >>
Jazz Musician >>
New Orleans Jazz >>
Windows Media >>
Musicians >>

Figure 2.4: Example of query expansion for "jazz" in Altavista.

2.3. Search results clustering

Cluster-based interface has been proven to be useful in navigation/exploration within large text collections [11]. Application of clustering to web search results can be beneficial in number of ways:

- the thematic organization of result set facilitate browsing/navigation through large result set
- proper labelling of clusters let user discovers main topics of the result set (automatic domain keywords acquisition) and can quickly identify interesting topic/cluster
- with results divided into categories, user can examine more relevant results (results that normally would be positioned far down in traditional ranked-list)
- user and can zoom-in/out specific categories to narrow results (which could be treated as a case of query expansion technique)

In the pioneering work [43] in web search results clustering, several keys requirements for successful clustering method have been stated:

Relevance clustering should separate relevant document from irrelevant ones

Browsable summaries, meaningful cluster description provide a concise, accurate description of cluster to ease user navigation

Ovelapping document with multiple topics can be assigned to multiple cluster

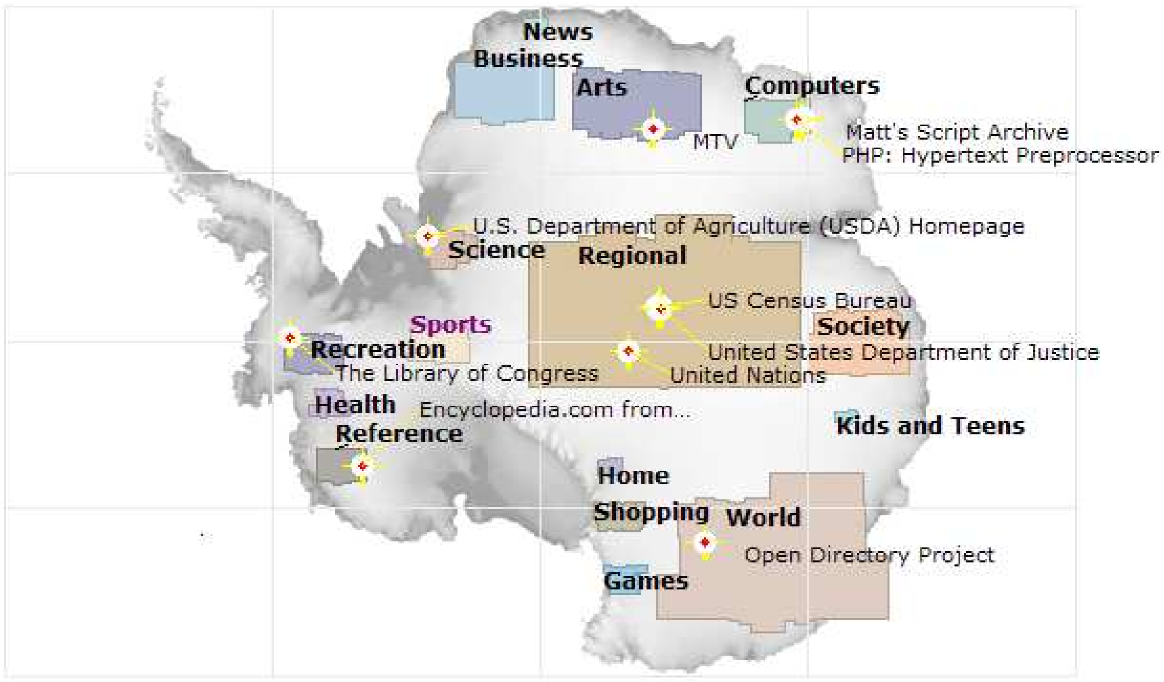


Figure 2.5: Sites listed in the Open Directory Project [24] organized within a map of Antartica [32]

Snippet tolerance methods should take into consideration that data provided in snippet is of limited length (thus not equal to the whole document it represents)

Speed reasonably fast, should be able to cluster 100-300 snippets in a few seconds

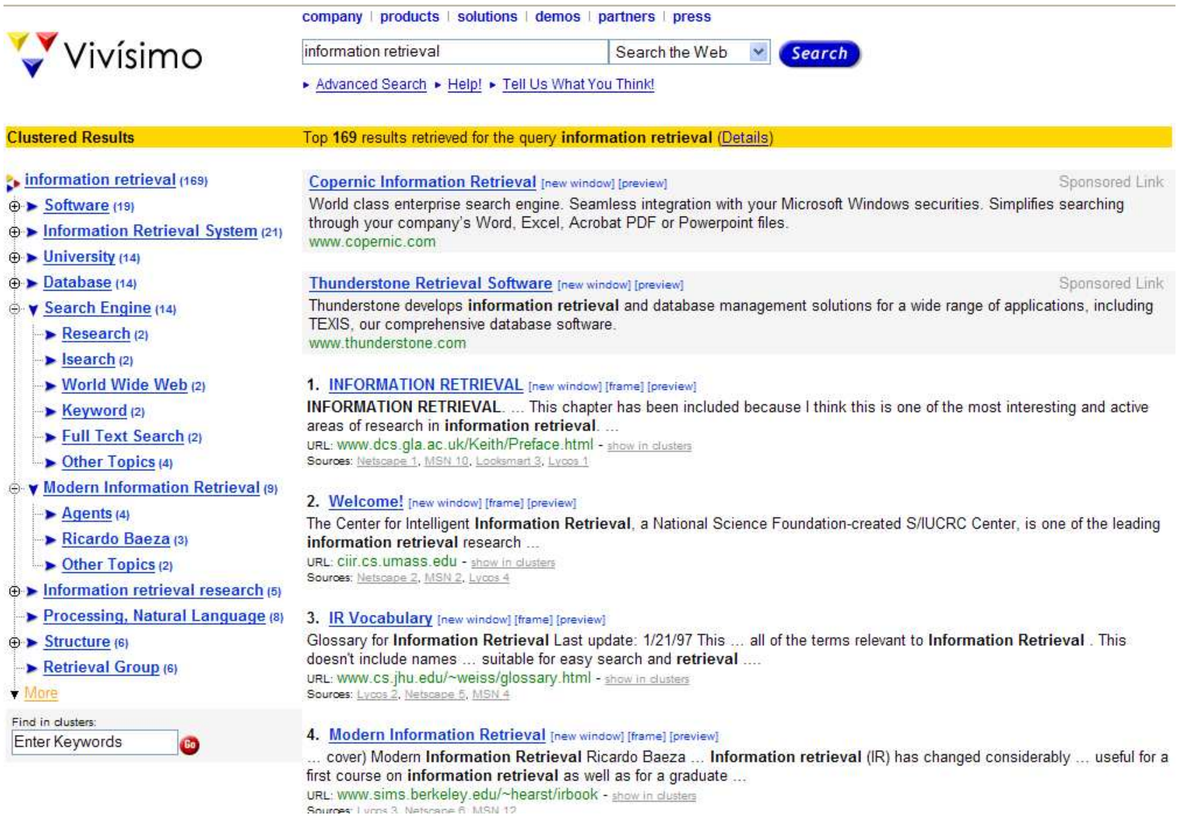
Incrementality processing of data should be started as soon as its first portion is received over the net

2.4. Previous work

There has been several approaches to the clustering results returned from information retrieval systems. Summary of most notable algorithms are presented below.

2.4.1. Scather/Gather

Scather/Gather originated as a cluster-based browsing method for large text collection [7]. It has been proven to communicate effective query terminology as well as topic structure of examined collection to user [22]. Adapted to organize post-retrieval documents [11], *Scather/Gather clusters documents into topically-coherent groups, and presents descriptive textual summaries to the user. The summaries consist of topical terms that characterize each cluster generally, and a number of typical titles that sample the contents of the cluster.* A non-hierarchical, partitioning strategy called Fractionation was used to cluster and re-cluster on-the-fly documents collection (or sub-collection formed by specific cluster) of size n to k groups in *O(kn)*. Documents are represented as vectors of terms, weighted using standard TF-IDF [5] scheme. Similarities between documents are measured by cosine of documents'



Clustered Results

- information retrieval (169)
  - Software (19)
  - Information Retrieval System (21)
  - University (14)
  - Database (14)
  - Search Engine (14)
    - Research (2)
    - Isearch (2)
    - World Wide Web (2)
    - Keyword (2)
    - Full Text Search (2)
    - Other Topics (4)
  - Modern Information Retrieval (9)
    - Agents (4)
    - Ricardo Baeza (3)
    - Other Topics (2)
  - Information retrieval research (5)
  - Processing, Natural Language (8)
  - Structure (6)
  - Retrieval Group (6)
  - More

Top **169** results retrieved for the query **information retrieval** (Details)

Copernic Information Retrieval [new window] [preview] — Sponsored Link
World class enterprise search engine. Seamless integration with your Microsoft Windows securities. Simplifies searching through your company's Word, Excel, Acrobat PDF or Powerpoint files.
www.copernic.com

Thunderstone Retrieval Software [new window] [preview] — Sponsored Link
Thunderstone develops **information retrieval** and database management solutions for a wide range of applications, including TEXIS, our comprehensive database software.
www.thunderstone.com

1. INFORMATION RETRIEVAL [new window] [frame] [preview]
INFORMATION RETRIEVAL. ... This chapter has been included because I think this is one of the most interesting and active areas of research in **information retrieval**. ...
URL: www.dcs.gla.ac.uk/Keith/Preface.html - show in clusters
Sources: Netscape 1, MSN 10, Looksmart 3, Lycos 1

2. Welcome! [new window] [frame] [preview]
The Center for Intelligent **Information Retrieval**, a National Science Foundation-created S/IUCRC Center, is one of the leading **information retrieval** research ...
URL: ciir.cs.umass.edu - show in clusters
Sources: Netscape 2, MSN 2, Lycos 4

3. IR Vocabulary [new window] [frame] [preview]
Glossary for **Information Retrieval** Last update: 1/21/97 This ... all of the terms relevant to **Information Retrieval** . This doesn't include names ... suitable for easy search and **retrieval** ....
URL: www.cs.jhu.edu/~weiss/glossary.html - show in clusters
Sources: Lycos 2, Netscape 5, MSN 4

4. Modern Information Retrieval [new window] [frame] [preview]
... cover) Modern **Information Retrieval** Ricardo Baeza ... **Information retrieval** (IR) has changed considerably ... useful for a first course on **information retrieval** as well as for a graduate ...
URL: www.sims.berkeley.edu/~hearst/irbook - show in clusters
Sources: Lycos 3, Netscape 6, MSN 12

Figure 2.6: Example of clustering web search results performed by Vivisimo [36].

vectors and given as input to clustering algorithm. Evaluation of Scather/Gather [11] approach shows that it can *provide significant improvement over similarity search ranking alone.*

2.4.2. Grouper

Grouper was the first post-retrieval system, designed specially for clustering web search results [43], [44]. It employed novel Suffix Tree Clustering (STC) algorithm to group together documents sharing common phrases (ordered sequence of words). This algorithm made use of special data structure called suffix tree (which is a kind of inverted index of phrases for a document collection) that can be constructed incrementally and linearly in time. Clustering is done by traversing the tree and merging nodes that share common document set. Clusters are then labelled using by the most notable phrases (which can be more informative than single words). This approach also allows overlapping clusters (document can be assigned into multiple clusters). Extended evaluation of Suffix Tree Clustering performed on whole document contents as well as document snippets shows it is at least comparable and on most case superior to classical methods like K-means, Hierarchical Agglomerative Clustering using vector space model.

2.4.3. Carrot, Carrot 2 Framework

Inspired by works on Grouper, an open-source implementation of Suffix Tree Clustering was developed [38], adding support for Polish language, and along with it, Carrot (now Carrot 2 [37]), *a research framework for experimenting with automated querying of various data sources* *(such as search engines), processing search results and their visualization.* Development of Carrot 2 framework has gavin momentum to research activities in search results clustering which led to creation of several new algorithms like LINGO [19], AHC [40].

2.4.4. Vivisimo

Vivisimo [36] is a commercial search result clustering engine. It has been known for its abilities to produce high quality hierarchical taxonomies from search results. The algorithm is proprietary thus no much detail is available. The official technology overview suggests that Vivisimo algorithm focus on forming hierarchies that best describe the results set:

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

2.4.5. SHOC, LINGO

LINGO [19] is based on similar idea like Vivisimo, which is "find cluster description first". It is inspired by SHOC [46], which harness Latent Semantic Indexing (LSI) to construct concept-document space from term-document space. Generally LINGO is composed of four phases. Preprocessing phase is first carried (which can cover various text filtering operation on original documents), follows by extraction of most frequent terms (words and phrases). Latent Semantic Indexing is next used to approximate term-document matrix, forming concept-document matrix. Labels for each concept are discovered by matching previously extracted terms that are closest to a concept by standard cosine measure. Each concept with labels forms a cluster, to which documents are assigned, based on how close they are to cluster label in the concept space. Due to LSI abilities to exploit of subtleties of term-document relation, labels produced by LINGO seem to well convey the cluster's contents. The speed of the algorithm depends primarily on complexity of Singular Value Decomposition (SVD) calculation for term-document matrix which can be computational expensive.

2.4.6. AHC

AHC [40] is an adaptation of Agglomerative Hierarchical Clustering algorithm to clustering search results. An n-gram extraction approach based on Smadja [28] was used to select phrases that form basis for documents representation. Hierarchical clustering is performed on vector space of documents and specials post-processing techniques are utilized to merge/prune clusters hierarchy on its labels.

2.4.7. CHCA

Worth mention for its simplicity is Class Hierarchy Construction Algorithm [1] which builds a hierarchy based only on binary version of term-document frequency matrix. The general idea is to create object-oriented like class hierarchy from document set, where each document could potentially become a class with terms as its attributes. Class that contains subset of attributes of another class becomes its parents. The process starts from most general class (document with least attributes) and continue until every document is assigned to a class.

Chapter 3

Document clustering

3.1. General definition

Clustering is an established and widely known technique for grouping data. It has been recognized and found successful applications in various areas like data mining [16, 8], statistics and information retrieval [5].

Let $D = \{d_1, d_2, \ldots, d_n\}$ be a set of objects, and $\delta(d_i, d_j)$ denote a similarity measure between objects d_i and d_j. *Clustering* then can be define as a task of finding the decomposition of D into K clusters $C = \{c_1, ..., c_k\}$ so that each object is assigned to a cluster and the ones belonging to the same cluster are similar to each other (regarding the similarity measure d), while as dissimilar as possible to objects from other clusters.

There are numerous clustering algorithms ranging from vector-space based, model-based (mixture resolving) to graph-theoretic spectral approaches. However, when concerning application to text, algorithms based on vector space are the most frequently used. In this work we will concentrate on vector space and provide a detail analysis of vector-based algorithms for document clustering. A readers interested in other clustering approaches is referred to [13, 16, 8].

3.1.1. Clustering in Information Retrieval

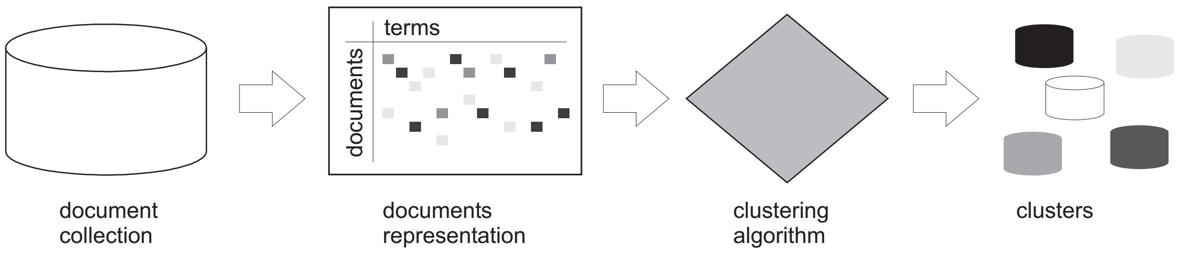


Figure 3.1: Document clustering process

While clustering has been used in various task of Information Retrieval (IR) [5, 6], it can be noticed that there are two main research themes in document clustering: as a tool to improve retrieval performance and as a way to organizing large collection of documents. Document clustering for retrieval purposes originates from the *Cluster Hypothesis* [35] which states that *closely associated documents tend to be relevant to the same requests.* By grouping *similar* documents together, one hopes that relevant documents will be separated from irrelevant ones, thus performance of retrieval in the clustered space can be improved. The second trend represented by [7, 11] found clustering to be a useful tool when browsing large collection of documents. Recently, it has been used in [44] [36] for grouping results returned from web search engine into thematically related cluster.

Several aspects need to be considered when approaching document clustering.

3.2. Vector space model and document representation

While in some domain such as data mining, objects of interest are frequently given in the form of feature/attributes vector, documents are given as sequences of words. Therefore, to be able to perform document clustering, an appropriate representation for document is needed. The most popular method is to represent documents as vectors in multidimensional space. Each dimension is equivalent to a distinct term (word) in the document collection. Due to the nature of text documents, the number of distinct terms (words) can be extremely large, counting in thousands for a relatively small to medium text collection. Computation in that high-dimensional space is prohibitively expensive and sometimes even impossible (e.g. memory size restriction). It is also obvious that not all words in the document are equally useful in describing its contents. Therefore, documents needs to be preprocessed to determine most appropriate terms for describing document semantic – *index terms*.

Assume that there are N documents $d_1, d_2, ..., d_N$, and M index terms enumerated from 1 to M. A document in vector space is represented by a vector :

$$d_i = [w_{i1}, w_{i2}, ..., w_{iM}]$$

where w_{ij} is a weight for the j-th term in document d_i.

3.2.1. Document preprocessing

Preprocessing document is an important task that can have great impact to the overall clustering performance [16, 13]. It can significantly shrinks the number of features (i.e. index terms) to be processed, thus reduces computational complexity, but also improve the quality and accuracy of selected terms. During the preprocessing phase, several operations on text can be performed to enhance the index terms selection.

Lexical analysis

Lexical analysis involves identification of words in the input document (given as a stream of characters). This mainly comes to breaking the text into word tokens, with regards to specified word separators. However, there are several cases that need to be treated with special concern such as digits, hyphens, punctuation marks and lower/upper case of the letters. For example, numbers are usually eliminated, as alone it doesn't seem to bring any meanings into document representation (except some special cases, e.g. retrieval on historical data constrained to specific dates). Punctuation marks (e.g. ".", "!", "?" etc.) usually can be removed (or specially marked to indicate sentence breaks) without affecting later processing phase, but it is not that obvious if a hyphenated phrase (e.g. "state-of-the-art") should be treated as a whole or broken up.

Stop-words list

Words that occur too frequently in documents of the collection are of little help in describing its contents. In the Web, for example, most documents will probably contains words "web", "site", "link", "www". A list of such words is called a *stop-words* list and can be used for discarding words from potential index terms. Words like "a", "the" (articles), "in" (prepositions), "and", "but" (conjunctions - connecting words), but also some popular verbs form ("be", "to"), adjectives and adverbs are usually included in the stop-words list.

Stemming

It is fact that many morphological variations of the words are usually related to the same semantic meaning. For example words "clusters", "clustering", "clustered" are related to the same central idea of "cluster". *Stemming* is the process of removing prefix or suffix of words, reducing to it root form (otherwise called *stem*). The use of stemming allows *conflating* word variations to a representative form thus reducing vocabulary size without reducing its semantic contents. Worth noticed is that stemming algorithms are usually based on simple rules for removing prefixes/suffixes and do not necessarily produce linguistically correct words (e.g. "computing", "computation" are reduced to "comput" while correct word would be "compute").

Doc	Title
d_1	Language modelling approach to information retrieval: the importance of a query term
d_2	Title language model for information retrieval
d_3	Two-stage language models for information retrieval
d_4	Building a web thesaurus from web link structure
d_5	Implicit link analysis for small web search
d_6	Query type classification for web document retrieval

Table 3.1: Example of document titles and its indexing terms (underlined words).

3.2.2. Weighting scheme

If index terms selection is viewed as a global way of discriminating meaningful terms for document description then introduction of weighting scheme can be viewed as a local approach to quantify the importance that each term bring into document representation. Weighting scheme can started as simple as **binary weighting** where an index terms is given weight 1 if it occurs in the document and 0 otherwise. More intuitive is a weighting scheme where terms are given weights that reflect its actual frequency in the document (i.e. the document is "more about" terms that occurs frequently in its contents) – **term frequency weighting**:

$$d_{ij} = tf_{ij} \quad \text{frequency of term } j \text{ in document } i$$

Term Frequency - Inverse Document Frequency weighting

The most frequently used weighting scheme is **TD*IDF** (term frequency - inverse document frequency) and its variations. The rationale behind TD*IDF is that terms that has high number of occurrences in a document (*tf factor*), are better characterization of document's semantic content than terms that occurs only a few times. However, terms that appears frequently in most documents in the collection will have little value in distinguishing document's content, thus the *idf factor* is used to downplay the role of terms that appears often in the whole collection.

$Document/Term$	$information$	web	$query$	$retrieval$	$model$	$language$
d_1	1	0	1	1	0	1
d_2	1	0	0	1	1	1
d_3	1	0	0	1	1	1
d_4	0	1	0	0	0	0
d_5	0	1	0	0	0	0
d_6	0	1	1	1	0	0

Table 3.2: Example of binary weighting for documents from 3.1

Let $t_1 \dots t_m$ denotes terms in the document corpus and $d_1 \dots d_n$ are documents in the corpus. In TD*IDF, the weight for each term t_j in document d_i is defined [25] as:

$$w_{ij} = tf_{ij} * log(n/df_j) \tag{3.1}$$

where tf_{ij} (term frequency, *tf*) - number of times term t_j occurs in document d_i, df_j (document frequency) - number of documents in the corpus in which term t_j occurs. The factor $log(N/df_j)$ is called *inverse document frequency (idf)* of term.

Figure 3.2: Plot of *inverse document frequency (idf)*

Document length normalization

As documents in the collection can be of various length, there might be a case when large documents can dominates the document space (components of its vectors have significant larger values than other document), causes clustering to produce skewed, incorrect solution. To overcome this problem, length normalization is usually applied to document's vector.

3.2.3. Similarity measure

The distance or similarity between document vectors is quantified by various measures that have been developed (see [25]):

Document/Term	*information*	*web*	*query*	*retrieval*	*model*	*language*
d_1	0.301	0	0.4771	0.1761	0	0.301
d_2	0.301	0	0	0.1761	0.4771	0.301
d_3	0.301	0	0	0.1761	0.4771	0.301
d_4	0	0.6021	0	0	0	0
d_5	0	0.301	0	0	0	0
d_6	0	0.301	0.4771	0.1761	0	0

Figure 3.3: Example of TF*IDF weighting for document from 3.1. Weights of document vector are normalized by its length.

Similarity Measure sim(X,Y)	**Evaluation of Binary Term Vectors**	**Evaluation of Weighted Term Vectors**
Inner product	$X \bigcap Y$	$\sum_{i=1}^{t} x_i \cdot y_i$
Dice coefficient	$2\frac{\|X \bigcap Y\|}{\|X\|+\|Y\|}$	$\frac{2\sum_{i=1}^{t} x_i y_i}{\sum_{i=1}^{t} x_i^2+\sum_{i=1}^{t} y_i^2}$
Cosine coefficient	$\frac{\|X \bigcap Y\|}{\|X\|^{1/2}+\|Y\|^{1/2}}$	$\frac{\sum_{i=1}^{t} x_i y_i}{\sqrt{\sum_{i=1}^{t} x_i^2+\sum_{i=1}^{t} y_i^2}}$
Jaccard coefficient	$\frac{\|X \bigcap Y\|}{\|X\|+\|Y\|-\|X \bigcap Y\|}$	$\frac{\sum_{i=1}^{t} x_i y_i}{\sum_{i=1}^{t} x_i^2+\sum_{i=1}^{t} y_i^2-\sum_{i=1}^{t} x_i y_i}$

Table 3.3: Measures of vector similarity [25]. X, Y are document vectors. x_i, y_i are the weight (value) of the i-th component of vector.

Due to its simplicity and intuitive interpretation, the cosine coefficient is most commonly used. When vectors representing documents are normalized by length, all above measures become quite similar to each other, ranging from 1 for identical documents, to 0 when there is nothing in common between documents.

3.2.4. Cluster representation

The choice of cluster representation is essential for clustering algorithm. Good representation of cluster should take into account not only features of documents belonging to cluster, but also should adequately quantify the degree that each feature contribute to cluster contents. In vector space based approach, a cluster is usually represented as mean vector (the centroid) of cluster documents. Given a cluster $C_k = \{d_{k1}, d_{k2}, ..., d_{km}\}$, we can define the *composite* vector D_k of cluster as:

$$D_k = \sum_{d \in C_k} d = d_{k1} + d_{k2} + ... + d_{km}$$

and the cluster's *centroid* as:

$$c_k = \frac{D_k}{|C_k|} = \frac{d_{k1} + d_{k2} + ... + d_{km}}{m}$$

Sometimes, a median — an object (document) closest to the centroid – chosen to represent a cluster. While clusters need not necessarily to be represented as vectors in the same space as documents, doing so greatly simplifies the similarity (or distance) calculation between two clusters or between a cluster and a document (the same similarity measure for documents can be used).

3.3. Clustering algorithms

Clustering algorithms based on vector space can be divided into two categories: hierarchical and partitioning.

3.3.1. Hierarchical algorithms

Agglomerative hierarchical clustering techniques starts with each object as a cluster, then iteratively merge two most similar clusters until only one cluster remains. As a result, the process creates a structure called dendogram - a tree which illustrates the merging process and intermediate clusters. With n objects there will be $n-1$ merging steps.

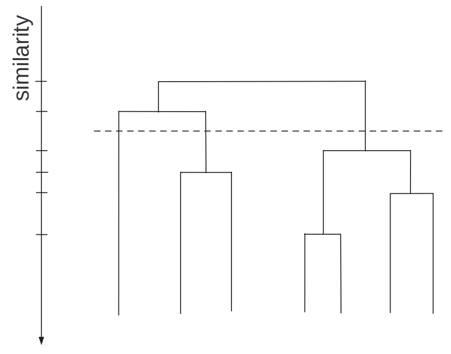


Figure 3.4: Dendogram from a hierarchical clustering solution. By cutting off at specified level of dendogram a partitional solution is obtained.

Algorithm 1 Agglomerative Hierarchical Clustering

```
Input: set of objects: $d_1, .., d_n$, similarity measures $\delta(d_i, d_j), i \neq j, i, j = 1..n$
1: create initial clusters from given set objects, each object forming a cluster
2: for $i = 1$ to $n - 1$ do
3:   find two most similar clusters and merge them creating a new cluster
4: end for
```

By using different measure for similarity between clusters, diverse results can be achieved:

single-linkage distance between two clusters is defined as the minimum distance between two objects, each coming from other cluster

complete-linkage opposite from single-linkage, distance between two clusters is defined as the maximum distance between two objects, one from each cluster

average-linkage the distance of two cluster is measured as an average of distances between all pairs of objects, where every pair consists of one object from each cluster

While complete-linkage algorithms are able to produce tight, compact clusters, more versatile solutions (clusters of different "shapes" e.g. concentric or elongated) can be achieved with single-linkage (but it also may results in a "chaining effect" – unnecessary merge due to existence of a chain of objects connecting two clusters [13]). In general, hierarchical algorithms are believed [13, 18] to achieve better clustering solutions than partitioning ones. However, more recently a comparison [31] shows that partitioning-based approach can be superior in document clustering.
Because the computational complexity of agglomerative hierarchical algorithms is $O(n^2 \log n)$ with memory consumption of $O(n^2)$ (for objects pair wise similarity matrix), where n – number of objects (documents), it is prohibited from application for large data set, where partitioning-based approach can be used.

3.3.2. Partitioning-based

Partitioning algorithm divide the set of n objects into k clusters so that the objective criterion is optimized. Objective criterion is usually chosen to minimize (maximize) some *similarity function*(e.g. distance) between objects. The most popular partitioning-based algorithm are *K-means* and its variants.

Algorithm 2 The *K-means* algorithm

```
Input: number of cluster $k$, set of $n$ objects
1: choose k objects arbitrarily to be representatives of $k$ initial clusters
2: repeat
3:     assign objects to closest cluster (in term of distance between given object and cluster's
       representative)
4:     recalculate clusters representatives
5: until no change or changes are small
```

The computation requirement of K-means is relatively low — $O(kn)$ (where k – number of cluster, n – number of documents) — hence it can be used for large collection of documents.

3.3.3. Optimization criteria

In partitioning-based approaches, the clustering is usually leads by a global optimization criterion. Some most frequently used criteria, see [47], are:

Maximize internal similarity

This criterion is based on the requirement that document within cluster must be similar to each other. Therefore, desired clustering solution should maximize the internal (within cluster) similarity between documents:

$$\mathcal{I}_1 = \sum_{r=1}^{k} n_r \left(\frac{1}{n_r^2} \sum_{d_i, d_j \in C_r} \delta(d_i, d_j)\right)$$

Maximize document-centroid similarity

Maximizing aggregated similarity between each document and its cluster centroid, is a very common [7, 18] criterion for vector-space K-means algorithms:

$$\mathcal{I}_2 = \sum_{r=1}^{k} \sum_{d_i \in C_r} \delta(d_i, c_r)$$

Minimize cluster-collection similarity

Based on the premise that documents in different cluster should be differentiate as much as possible (i.e. documents share very few terms across clusters), one may come to a criterion to make centroid of clusters as mutually orthogonal as possible. However, [47] pointed out that for this criterion, trivial solutions could be achieved (e.g. $k-1$ clusters, each contains a single document that share few terms with the rest, and the remaining documents are assigned to single cluster). Thus, a criterion, that aims to stretch out clusters by minimizing its similarity to central point (centroid) of the collection, has been proposed [47]:

$$\mathcal{E}_1 = \sum_{r=1}^{k} n_r \delta(C_r, C)$$

where C – centroid of the whole collection.

3.3.4. Hard vs. soft assignment

For some application, especially with text, when assigning document to cluster, one may want to quantify how "related" is the document to a cluster by associating real value – the membership degree. In document clustering, if we view clusters as a set of document sharing similar topic/theme, then it is natural that one document can be "about" several topics anbd could be assigned to several clusters. The algorithm that uses the above notion is said to use fuzzy/soft assignment with overlapping clusters. In opposite, algorithm with hard assignment allows each document to belong only to one cluster without quantifying degree of the relation.

Chapter 4

Rough Sets and the Tolerance Rough Set Model

Rough sets theory was originally developed [20, 14] as a tool for data analysis and classification. It has been successfully applied in various tasks, such as feature selection/extraction, rule synthesis and classification [14]. In this chapter we will present fundamental concepts of rough sets with illustrative examples. Some extensions of rough set are described, concentrating on the use of rough set to synthesize approximations of concepts from data.

4.1. Approximation of concept

Consider a non-empty set of object U called the universe. Suppose we want to define a concept over universe of objects U. Let assume that our concept can be represented as subset X of U. The central point of rough set theory is the notion of set approximation: any set in U can be approximated by its *lower* and *upper approximation*.

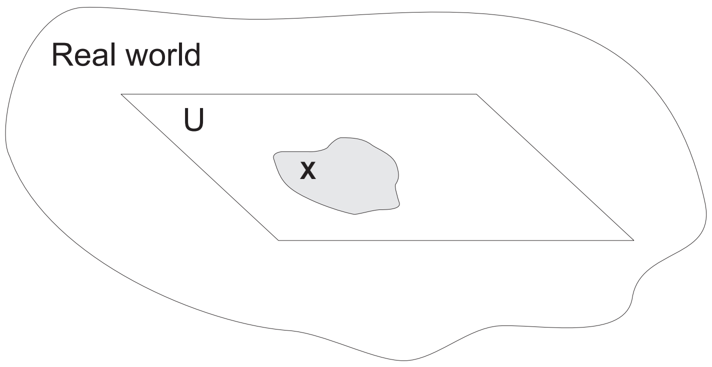


Figure 4.1: Rough concept

4.1.1. Indiscernibility relation

Originally [20], in order to define lower and upper approximation we need to introduce an *indiscernibility relation*: $R \subseteq U \times U$ (where R can be any equivalence relation). For two objects $x, y \in U$, if xRy then we say that x and y are *indiscernible* from each other. The indiscernibility relation R induces a complete partition of universe U into equivalent classes $[x]_R$, $x \in U$.

We define lower and upper approximation of set X, with regards to an *approximation space* denoted by $\mathcal{A} = (U, R)$, respectively as:

$$\begin{aligned} \mathbf{L}_R(X) &= \{x \in U : [x]_R \subseteq X\} \\ \mathbf{U}_R(X) &= \{x \in U : [x]_R \cap X \neq \emptyset\} \end{aligned} \tag{4.1}$$

Due to the fact that $\mathbf{L}_R \subseteq X \subseteq \mathbf{U}_R$, there is a special set $\mathbf{BN}_R = \mathbf{U}_R - \mathbf{L}_R$, called a *boundary region of approximation.* Intuitively,F lower approximation contains objects that **certainly** belong to our concept while upper approximation contains objects that **may** belong to our concept. Together, the pair $(\mathbf{L}_R, \mathbf{U}_R)$ constitutes for a *rough approximation* (or *rough set*) of concept X.

4.1.2. Rough membership

Approximations can also be defined by mean of *rough membership function* [45]. Given rough membership function $\mu_X : U \to [0, 1]$ of a set $X \subseteq U$, the rough approximation is defined as

$$\begin{aligned} \mathbf{L}_\mu(X) &= \{x \in U : \mu(x, X) = 1\} \\ \mathbf{U}_\mu(X) &= \{x \in U : \mu(x, X) > 0\} \end{aligned} \tag{4.2}$$

Note that, given rough membership function as $\mu_X(x, X) = \frac{|[x]_R \cap X|}{|[x]_R|}$, this definition is actually equivalent to 4.1.

4.1.3. Information Systems

Until now, objects in the universe U were regarded as something abstract. In practice however, objects are usually real-world entities that are described by some information (most often denoted as $inf(x)$ - information about object x). This is captured in a notion of *information system.* Formally, an information system is described as a pair:

$$\mathcal{I} = (U, A)$$

where U - non-empty, finite set of *objects* called the *universe*, A - non-empty, finite set of *attributes.* For each attribute $a \in A$ there is a corresponding *evaluation function* $f_a : U \to V_a$, where V_a is the *value set* of a. Intuitively any finite set of objects, in which objects are described by set of attributes can be perceived as an information system, e.g.: a group of people, where each person is characterized by their sex, age and occupation. The simplest form of information system is information table, a two dimensional array where rows correspond to objects and columns to attributes. For each object $x \in U$, the perceived information about x with w.r.t. set of attributes $B \subseteq A$ is defined as *B-information vector*:

$$inf_B(x) = \{(a, f_a(x)) : a \in B\}$$

Frequently, the information table is given in an extended form — with one additional distinguish column that contains a decision — which is referred as *decision table.*

Example 1 *Let consider an universe of hospital patients* $U = \{p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8\}$*,given in a decision table 4.1 [21]. Patients are described by their symptoms of sickness.*

Let an equivalence relation R be defined by equality of two attributes Headache and Temperature (i.e. $xRy \Leftrightarrow f_{Headache}(x) = f_{Headache}(y) \wedge f_{Temperature}(x) = f_{Temperature}(y)$*). This*

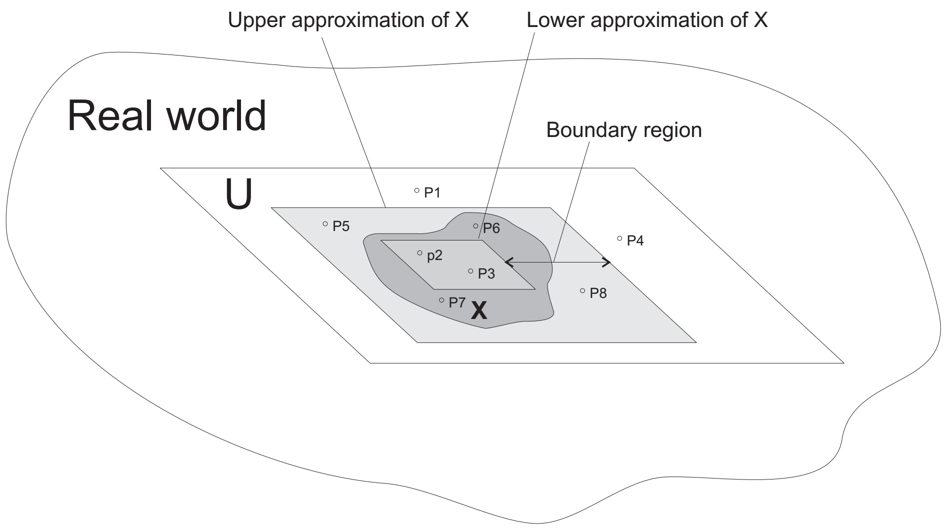


Figure 4.2: Visualization of lower, upper approximation and boundary region of example set X in the universe U.

	Attributes			**Decision**
Patient	Headache	Muscle pain	Temperature	Flu
p_1	yes	yes	normal	no
p_2	yes	yes	high	yes
p_3	yes	yes	very high	yes
p_4	no	yes	normal	no
p_5	no	no	high	no
p_6	no	yes	very high	yes
p_7	no	no	high	yes
p_8	no	yes	very high	no

Table 4.1: Example of an information system.

equivalence relation induces a partition of U into classes $\{p_1\}, \{p_2\}, \{p_3\}, \{p_4\}, \{p_5, p_7\}, \{p_6, p_8\}$. We can see for example, that p_5 and p_7 are indiscernible with regards to relation R. Assuming that the concept X we want to describe is given here as a decision — if a patient has flu ($X = \{p_2, p_3, p_6, p_7\}$) — its approximation can be determined (Figure 4.2):

$$\mathbf{L}_R(X) = \{p_2, p_3\}$$

$$\mathbf{U}_R(X) = \{p_2, p_3, p_5, p_6, p_7, p_8\}$$

4.2. Generalized approximation spaces

The classical rough set theory is based on equivalence relation that divides the universe of objects into disjoint classes. By definition, an equivalence relation $R \subseteq U \times U$ is required to be:

1. reflexive: xRx, for any $x \in U$
2. symmetric: $xRy \Rightarrow yRx$, for any $x, y \in U$

3. transitive: $xRy \wedge yRz \Rightarrow xRz$, for any $x, y, z \in U$

Practically, for some applications, the requirement for equivalent relation has showed to be too strict. The nature of the concepts in many domains are imprecise and can be overlapped additionally.

Example 2 *Let us consider a collection of scientific documents and keywords describing those documents. It is clear that each document can have several keywords and a keyword can be associated with many documents. Thus, in the universe of documents, keywords can form overlapping classes.*

By relaxing the relation R to a tolerance relation, where transitivity property is not required, a generalized tolerance space is introduced by Skowron [2].

4.2.1. Definition

Generalized approximation space is defined as a quadruple

$$\mathcal{A} = (U, I, \nu, P)$$

where

1. U is a non-empty set of objects (an universe)
2. $I : U \to \mathcal{P}(U)$ is an *uncertainty function* ($\mathcal{P}(U)$ - set of all subsets of U)
   An uncertainty function I can be any function satisfying conditions:
   - $x \in I(x)$ for $x \in U$
   - $y \in I(x) \iff x \in I(y)$ for any $x, y \in U$.

   thus the relation $xRy \iff y \in I(x)$ is a tolerance relation (i.e. reflexive, symmetric) and $I(x)$ is a tolerance class of x. Intuitively, if we consider that objects $x \in U$ are viewed through "lenses" of available information $inf(x)$ then $I(x)$ is the set of objects in U that "look" *similar* to x.
3. $\nu : \mathcal{P}(U) \times \mathcal{P}(U) \to [0, 1]$ is a *vague inclusion function*
   Vague inclusion ν is a kind of membership function (defined in 4) but extended to functions over $\mathcal{P}(U) \times \mathcal{P}(U)$ to measure degree of inclusion between two set. Vague inclusion can be any function

   $$\nu : \mathcal{P}(U) \times \mathcal{P}(U) \to [0, 1]$$

   that is *monotonic* with respect to the second argument:

   $$Y \subseteq Z \;\Rightarrow\; \nu(X, Y) \leq \nu(X, Z) \quad \text{for} \;\; X, Y, Z \subseteq U$$

   Together with uncertainty function I, vague inclusion function ν defines the *rough membership function* for $x \in U, X \subseteq U$:

   $$\mu_{I,\nu}(x, X) = \nu(I(x), X)$$
4. $P : I(U) \to \{0, 1\}$ is a *structurality function*
   The introduction of structurality function $P : I(U) \to \{0, 1\}$ ($I(U) = \{I(x) : x \in U\}$) allows us to enforce additional global conditions on sets $I(x)$ considered to be approximated. In generation of approximations, only sets $X \in I(U)$ for which $P(X) = 1$ (referred as *P-structural element* in U) are considered. For example, function $P_\alpha(X) = 1 \iff |X|/|U| > \alpha$ will discard all subsets that are relatively smaller than certain percentage (given by α) of U.

4.2.2. Approximations

Approximations in $\mathcal{A}$ of any $X \subseteq U$ are then defined as

$$\mathbf{L}_{\mathcal{A}}(X) = \{x \in U : P(I(X)) = 1 \ \wedge \nu(I(x), X) = 1\} \tag{4.3}$$
$$\mathbf{U}_{\mathcal{A}}(X) = \{x \in U : P(I(X)) = 1 \ \wedge \nu(I(x), X) > 0\} \tag{4.4}$$

With given definition above, generalized approximation spaces can be used in any application where I, ν and P can be appropriately determined.

Example 3 *The standard rough set model (4.1) can be described as a particular case of generalized approximation. Let consider a quadruple $\mathcal{A}_1 = (U, I, \nu, P)$ where*

- *U – universe of objects*
- *$I(x) = [x]_R$ where $R \subseteq U \times U$ is an indiscernibility relation*
- *$\nu(X, Y) = \frac{|X \cap Y|}{|Y|}$ where $X, Y \subseteq U$*
- *$P(I(x)) = 1$ for every $x \in U$*

With $P(I(x)) = 1$ for every x, the approximations can be simplified to:

$$\mathbf{L}_{\mathcal{A}_1}(X) = \{x \in U : \nu(I(x), X) = 1\} \tag{4.5}$$
$$\mathbf{U}_{\mathcal{A}_1}(X) = \{x \in U : \nu(I(x), X) > 0\} \tag{4.6}$$

One can find that this is equivalent to the definition 4.2 where ν is a kind of rough membership function.

Example 4 Variable Precision Rough Set *[48] proposed a generalization of approximations by introducing a special non-decreasing function $f_\beta : [0,1] \to [0,1]$ (for $0 \leq \beta < 0.5$) satisfying properties:*

- *$f_\beta(t) = 0 \iff 0 \leq t \leq \beta$*
- *$f_\beta(t) = 1 \iff 1 - \beta \leq t \leq 1$*

The generalized membership function called f_β-membership function is next defined as

$$\mu(f_\beta)(x, X) = f_\beta(t) = f_\beta(\frac{|[x]_R \cap X|}{|[x]_R|})$$

The lower and upper approximations of a set $X \subseteq U$ with respect to $\mu(f_\beta)$ can be defined as follows:

$$\mathbf{L}_{\mu(f_\beta)}(X) = \{x \in U : \mu(x, X) = 1\} \tag{4.7}$$
$$\mathbf{U}_{\mu(f_\beta)}(X) = \{x \in U : \mu(x, X) > 0\} \tag{4.8}$$

With $\beta = 0$ and f_β equal to identity on $[0,1]$ we have the case of classical rough set [20].

By varying the f_β, the extension introduced by variable precision rough set model can be viewed as a method of controlling the size and gradualness of boundary region.

Variable Precision Rough Set can be also modelled as a generalized approximation space $\mathcal{A}_\beta = (U, I, \nu, P)$ with

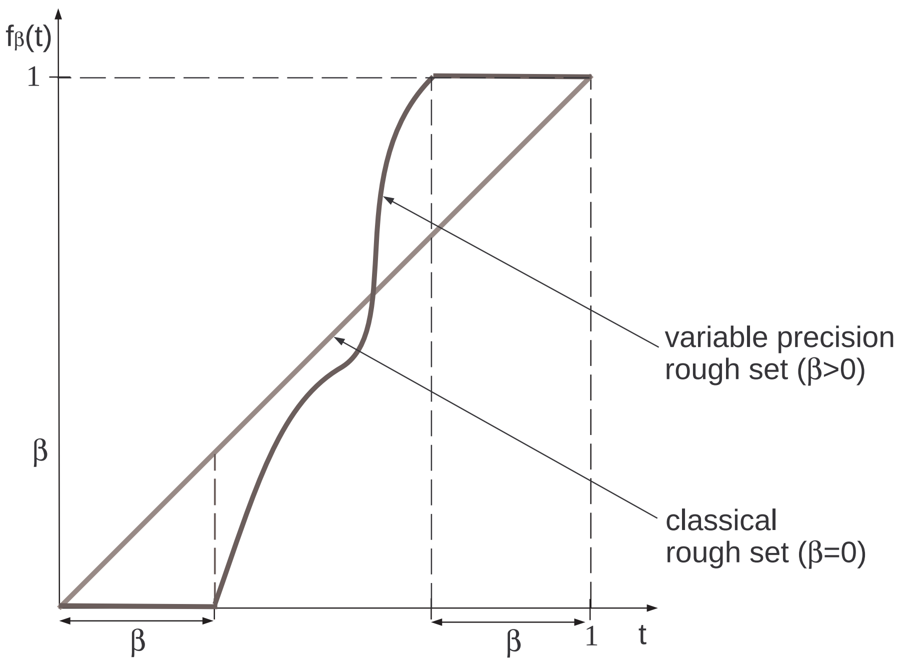


Figure 4.3: Example of f_β for variable precision rough set ($\beta > 0$) and classical rough set ($\beta = 0$)

- *U – universe of objects*
- *$I(x) = [x]_R$ where $R \subseteq U \times U$ is an indiscernibility relation*
- *$\nu(X, Y) = f_\beta(\frac{|X \cap Y|}{|Y|})$ where $X, Y \subseteq U$*
- *$P(I(x)) = 1$ for every $x \in U$*

4.3. Tolerance Rough Set Model

Tolerance Rough Set Model (TRSM) was developed [26, 34] as basis to model documents and terms in information retrieval, text mining, etc. With its ability to deal with vagueness and fuzziness, tolerance rough set seems to be promising tool to model relations between terms and documents. In many information retrieval problems, especially in document clustering, defining the relation(i.e. similarity or distance) between document-document, term-term or term-document is essential. In Vector Space Model, is has been noticed [34] that a single document is usually represented by relatively few terms[1]. This results in zero-valued similarities which decreases quality of clustering. The application of TRSM in document clustering was proposed as a way to enrich document and cluster representation with the hope of increasing clustering performance.

[1]In other words, number of non-zero values in document's vector is much smaller than vector's dimension – the number of all index terms

4.3.1. Tolerance space of terms

Let $D = \{d_1, d_2, \dots, d_N\}$ be a set of document and $T = \{t_1, t_2, \dots, t_M\}$ set of *index terms* for D. With the adoption of Vector Space Model (3.2) each document d_i is represented by a weight vector $[w_{i1}, w_{i2}, \dots, w_{iM}]$ where w_{ij} denoted the weight of term j in document i. In TRSM, the tolerance space is defined over a universe of all index terms

$$U = T = \{t_1, t_2, \dots, t_M\}$$

The idea is to capture conceptually related index terms into classes. For this purpose, the tolerance relation R is determined as the co-occurrence of index terms in all documents from D. The choice of co-occurrence of index terms to define tolerance relation is motivated by its meaningful interpretation of the semantic relation in context of IR and its relatively simple and efficient computation.

Tolerance class of term

Let $f_D(t_i, t_j)$ denotes the number of documents in D in which both terms t_i and t_j occurs. The uncertainty function I with regards to threshold θ is defined as

$$I_\theta(t_i) = \{t_j \mid f_D(t_i, t_j) \geq \theta\} \cup \{t_i\}$$

Clearly, the above function satisfies conditions of being reflexive: $t_i \in I_\theta(t_i)$ and symmetric: $t_j \in I_\theta(t_i) \iff t_i \in I_\theta(t_j)$ for any $t_i, t_j \in T$. Thus, the tolerance relation $\mathcal{I} \subseteq T \times T$ can be defined by means of function I:

$$t_i \mathcal{I} t_j \iff t_j \in I_\theta(t_i)$$

where $I_\theta(t_i)$ is the *tolerance class* of index term t_i.

In context of Information Retrieval, a tolerance class represents a concept that is characterized by terms it contains. By varying the threshold θ (e.g. relatively to the size of document collection), one can control the degree of relatedness of words in tolerance classes (or in other words the preciseness of the concept represented by a tolerance class).

To measure degree of inclusion of one set in another, vague inclusion function is defined as

$$\nu(X, Y) = \frac{|X \cap Y|}{|X|}$$

It is clear that this function is monotonous with respect to the second argument. The membership function μ for $t_i \in T$, $X \subseteq T$ is then defined as

$$\mu(t_i, X) = \nu(I_\theta(t_i), X) = \frac{|I_\theta(t_i) \cap X|}{|I_\theta(t_i)|}$$

With the assumption that the set of index terms T doesn't change in the application, all tolerance classes of terms are considered as structural subsets: $P(I_\theta(t_i)) = 1$ for all $t_i \in T$.

Finally, the lower and upper approximations of any subset $X \subseteq T$ can be determined — with the obtained tolerance $\mathcal{R} = (T, I, \nu, P)$ — respectively as

$$\mathbf{L}_\mathcal{R}(X) = \{t_i \in T \mid \nu(I_\theta(t_i), X) = 1\}$$

$$\mathbf{U}_\mathcal{R}(X) = \{t_i \in T \mid \nu(I_\theta(t_i), X) > 0\}$$

One interpretations of the given approximations can be as follows: if we treat X as an concept described vaguely by index terms it contains, then $\mathbf{U}_\mathcal{R}(X)$ is the set of concepts that share some semantic meanings with X, while $\mathbf{L}_\mathcal{R}(X)$ is a "core" concept of X.

Example 5 *Consider an universe of unique terms extracted from a set of search result snippets returned from Google search engine for a "famous"*[2] *query:* **jaguar**. *Tolerance classes are generated for threshold $\theta = 9$. It is interesting to observe that the generated classes do reveal different meanings of the word "jaguar": a cat, a car, a game console, an operating system and some more.*

Term	Tolerance class	Document frequency[3]
Atari	**Atari, Jaguar**	**10**
Mac	**Mac, Jaguar, OS, X**	**12**
onca	onca, Jaguar, Panthera	9
club	Jaguar,club	27
Panthera	**onca, Jaguar, Panthera**	**9**
new	Jaguar, new	29
information	Jaguar, information	9
OS	Mac,Jaguar, OS, X	15
site	Jaguar, site	19
Welcome	Jaguar, Welcome	21
X	Mac, Jaguar, OS, X	14
Cars	**Jaguar, Cars**	**24**

Table 4.2: Tolerance classes of terms generated from 200 snippets return by Google search engine for a query "jaguar" with $\theta = 9$; only non-trivial classes are shown.

4.3.2. Enriching document representation

In standard Vector Space Model, a document is viewed as a *bag of words/terms*. This is articulated by assigning, non-zero weight values, in document's vector, to terms that occurs in document. With TRSM, the aim is enrich representation of document by taking into consideration not only terms actually occurring document but also other related terms with similar meanings. A "richer" representation of document can be acquired by representing document as set of tolerance classes of terms it contains. This is achieved by simply representing document with its upper approximation. Let $d_i = \{t_{i_1}, t_{i_2}, ..., t_{i_k}\}$ be a document in D and $t_{i_1}, t_{i_2}, ..., t_{i_k} \in T$ are index terms of d_i:

$$\mathbf{U}_{\mathcal{R}}(d_i) = \{t_i \in T \mid \nu(I_\theta(t_i), d_i) > 0\}$$

4.3.3. Extended weighting scheme for upper approximation

To assign weight values for document's vector, the TF*IDF (see 3.2.2) weighting scheme is used. In order to employ approximations for document, the weighting scheme need to be extended to handle terms that occurs in document's upper approximation but not in the document itself (or terms that occurs in the document but not in document's lower

[2]The word **jaguar** is frequently used as a test query in information retrieval (e.g. document classification, clustering, word disambiguation) because it is a polysemy (i.e. word that has several meanings) especially in the web: jaguar as a cat (*panthera onca* - http://dspace.dial.pipex.com/agarman/jaguar.htm); as an Jaguar car; it was a name for a game console made by Atari - http://www.atari-jaguar64.de, but also a codename for Apple's newest operating system MacOS X - http://www.apple.com/macosx

approximation). The extended weighting scheme is defined as:

$$w_{ij} = \begin{cases} (1 + log(f_{d_i}(t_j)) * \log \frac{N}{f_D(t_j)} & \text{if } t_j \in d_i \\ \min_{t_k \in d_i} w_{ik} * \frac{\log \frac{N}{f_D(t_j)}}{1+\log \frac{N}{f_D(t_j)}} & \text{if } t_j in \mathbf{U}_{\mathcal{R}}(d_i) \backslash d_i \\ 0 & \text{if } t_j \notin in \mathbf{U}_{\mathcal{R}}(d_i) \end{cases}$$

(where w_{ij} is the weight for term t_j in document d_i). The extension ensures that each terms occurring in upper approximation of d_i but not in d_i, has a weight smaller than the weight of any terms in d_i. Normalization by vector's length is then applied to all document vectors:

$$w_{ij} = \frac{w_{ij}}{\sqrt{\sum_{t_k \in d_i} (w_{ij})^2}}$$

TITLE	EconPapers: Rough sets bankruptcy prediction models versus auditor
DESCRIPTION	Rough sets bankruptcy prediction models versus auditor signalling rates. Journal of Forecasting, 2003, vol. 22, issue 8, pages 569-586. Thomas E. McKee. ...

Document vector			
original		using upper approximation	
Term	Weight	Term	Weight
auditor	0.567	auditor	0.564
bankruptcy	0.4218	bankruptcy	0.4196
signalling	0.2835	signalling	0.282
EconPapers	0.2835	EconPapers	0.282
rates	0.2835	rates	0.282
versus	0.223	versus	0.2218
issue	0.223	issue	0.2218
Journal	0.223	Journal	0.2218
MODEL	0.223	MODEL	0.2218
prediction	0.1772	prediction	0.1762
Vol	0.1709	Vol	0.1699
		applications	0.0809
		Computing	0.0643

Table 4.3: Example snippet and its upper approximation.

Chapter 5

The Tolerance Rough Set clustering algorithm for search results

This chapter starts with the introduction of document clustering algorithms based on Tolerance Rough Set Model. The chapter continues by outlining special characteristic of data which is search results. An adaptation of TRSM-based algorithm for web search results clustering is next proposed and described in details.

5.1. Document clustering algorithms based on TRSM

With the introduction of Tolerance Rough Set Model in [26], several document clustering algorithms based on that model was also introduced [26, 34].
The main novelty that TRSM brings into clustering algorithms are the way of representing clusters and document.

5.1.1. Cluster representation

As discussed earlier in 3.2.4, determining cluster representation is very important factor in partitioning-based clustering. Frequently, cluster is represented as a mean or median of all documents it contains. Sometime however, a representation not based on vector is needed as cluster description is directly derived from its representation. For example, cluster can be represented by most "distinctive" terms from cluster's documents (e.g. most frequent terms; or frequent terms in clusters but infrequent globally).

In [34], an approach to construct a *polythetic* representation is presented. Let R_k denotes a representative for cluster k. The aim is to construct a set of index terms R_k representing cluster C_k so that:

- each document d_i in C_k share some or many terms with R_k
- terms in R_k occurs in most documents in C_k
- terms in R_k needs not to be contained by every document in C_k

The weighting for terms t_j in R_k is calculated as an averaged weight of all occurrences in documents of C_k:

$$w_{kj} = \frac{\sum_{d_i \in C_k} w_{ij}}{|\{d_i \in C_k \mid t_j \in d_i\}|}$$

Algorithm 3 Determine cluster representatives

```
1: $R_k = \emptyset$
2: for all $d_i \in C_k$ and $t_j \in d_i$ do
3:    if $f_{C_k}(t_j)/|C_k| > \sigma$ then
4:       $R_k = R_k \cup t_j$
5:    end if
6: end for
7: if $d_i \in C_k$ and $d_i \cap R_k = \emptyset$ then
8:    $R_k = R_k \cup \text{argmax}_{t_j \in d_i} w_{ij}$
9: end if
```

Let $f_{C_k}(t_J)$ be the number of documents in C_k that contain t_j. The above assumptions lead to following rules to create cluster representatives:

To the initially empty representative set, terms that occur frequent enough (controlled by threshold σ) in documents within cluster are added. After this phase, for each document that is not yet "represented" in representative set (i.e. document shares no terms with R_k), the strongest/heaviest term from that document is added to the cluster representatives.

5.1.2. TRSM-based non-hierarchical clustering algorithm

The non-hierarchical document clustering algorithm based on TRSM is a variation of a K-means clustering algorithm (3.3.2) with overlapping clusters. The modifications introduced are:

- use of document's upper approximation when calculating document-cluster and document-document similarity
- documents are soft-assigned to cluster with associated membership value
- use nearest-neighbor to assign unclassified documents to cluster

The use of upper approximation in similarity calculation to reduce the number of zero-valued similarities is the main advantage main advantage TRSM-based algorithms claimed to have over traditional approaches. This allows the situation, in which two document are similar (i.e. have non-zero similarity) although they do not share any terms.

5.1.3. TRSM-based hierarchical clustering algorithm

A hierarchical agglomerative clustering algorithm based on TRSM was proposed in [26]. It utilizes upper approximation to calculate cluster similarities in the merging step.

Both presented clustering algorithms were evaluated with standard test collections and has shown some success [26, 34].

5.2. Clustering search results

Despite being derived from document clustering, methods for clustering web search results differ from its ancestors on numerous aspects. Most notably, document clustering algorithms are designed to works on relatively large collection of full-text document (or sometimes document abstract). In opposite, the algorithm for web search results clustering is supposed to works on moderate size (several hundreds elements) set of text snippets (with length of

Algorithm 4 TRSM-based non-hierarchical clustering algorithm

```
Input: $D$ – set of $N$ documents, $K$ – number of clusters
Output: $K$ overlapping clusters of documents from $D$ with associated membership value
 1: Randomly select $K$ document from $D$ to serve as $K$ initial cluster $C_1, C_2, ..., C_K$. Each
    chosen document also forms cluster representative $R_1, R_2, ..., R_K$.
 2: repeat
 3:   for each $d_i \in D$ do
 4:     for each cluster $C_k$, $k = 1, .., K$ do
 5:       calculate the similarity between document's upper approximation and the cluster
          representatives $S(\mathbf{U}_{\mathrm{R}}(d_i), R_k)$
 6:       if $S(\mathbf{U}_{\mathrm{R}}(d_i), R_k) > \delta$ then
 7:         assign $d_i$ to $C_k$ with the similarity taken as cluster membership: $m(d_i, C_k) =$
            $S(\mathbf{U}_{\mathrm{R}}(d_i), R_k)$
 8:       end if
 9:     end for
10:   end for
11:   for each cluster $C_k$ do
12:     re-determine cluster representative $R_k$
13:   end for
14: until the is little or no change in cluster membership
15: for each unclassified document $d_u$ (document has not been assigned to any cluster) do
16:   let $NN(d_u)$ denotes nearest neighbor (with non-zero similarity) of $d_u$
17:   assign $d_u$ to cluster $C_k$ that contains $NN(d_u)$
18:   calculate membership $m(d_u, C_k) = m(NN(d_u), C_k) \cdot S(\mathbf{U}_{\mathrm{R}}(d_u), \mathbf{U}_{\mathrm{R}}(NN(d_u)))$
19: end for
20: for each cluster $C_k$ do
21:   re-determine cluster representative $R_k$
22: end for
```

Algorithm 5 TRSM-based hierarchial agglomerative clustering algorithm

```
Input: $D$ – set of $N$ documents, $S$ – similarity measure
Output: hierarchical structure $H$ of clusters $C_k$
 1: Initially, each document $d_i \in D$ form a one document cluster $C_i = d_i$. $H = C_1, C_2, ..., C_N$.
 2: while there is more than one cluster, $|H| > 1$ do
 3:   find two most similar clusters in terms of their representative upper approximation
      $(C_{n1}, C_{n2}) = \text{argmax}_{(C_u, C_v) \in H \times H} S(\mathbf{U}_{\mathrm{R}}(R_u), \mathbf{U}_{\mathrm{R}}(R_v))$
 4:   merge two cluster $C_{n1} \cup C_{n2} = C_k$ and let $H = (H \ \{C_{n1}, C_{n2}\}) \cup \{C_k\}$
 5: end while
```

10-20 words). In document clustering, the main emphasis is put on the quality of clusters and the scalability to large number of documents, as it is usually used to process the whole document collection (e.g. for document retrieval on clustered collection). For web search results clustering, apart from delivering good quality clusters, it is also required to produce meaningful, concise description for cluster. Additionally, the algorithm must be extremely fast to process results on-line (as post-processing search results before delivered to the user) and scale with the increase of user requests (e.g. measures as number of processed requests per specified amount of time).

document clustering	**web search results clustering**
full-text documents (or document abstract)	short text snippets
off-line processing of large collection	on-line processing of moderate size set
cluster quality	cluster quality + cluster description meaningfulness
scalability with number of documents	scalability with number of user requests

Table 5.1: Comparison between document clustering and web search results clustering

5.3. The TRC algorithm

The Tolerance Rough Clustering algorithm is based primarily on the K-means algorithm presented in (5.1.2) [34]. By adapting K-means clustering method, the algorithm remain relatively quick (which is essential for on-line results post-processing, see 2.3) while still maintaining good clusters quality. The usage of Tolerance Space and upper approximation to enrich inter-document and document-cluster relation allows the algorithm to discover subtle similarities not detected otherwise. As it has been mentioned, in search results clustering, the proper labelling of cluster is as important as cluster contents quality. Since the use of phrases in cluster label has been proven [44, 19] to be more effective than single words, TRC algorithm utilize n-gram of words (phrases) retrieved from documents inside cluster as candidates for cluster description.

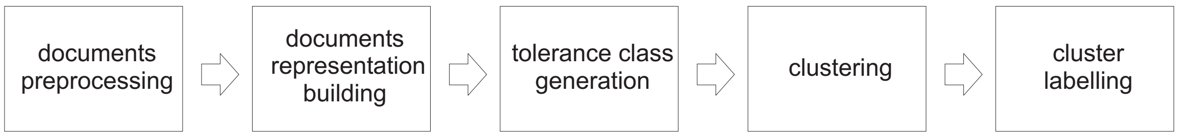


Figure 5.1: Phases of TRC algorithm

The TRC algorithm is composed from five phases (depicted in 5.1):

1. documents preprocessing
2. documents representation building
3. tolerance class generation
4. clustering
5. cluster labelling

5.3.1. Preprocessing

It is widely known [16] that preprocessing text data before feeding it into clustering algorithm is essentials and can have great impact on algorithm performance. In TRC, several preprocessing steps are performed on snippets:

Text cleansing

In this step, text content of snippet is cleaned from unusable terms such as:

- non-letter characters like , $, #
- HTML-related tags (e.g.= `<a>`, `<p>`) and entities (e.g. &amp, &quot)

Stemming

A version of Porter's stemmer [23] is used in this step to remove prefixes and suffixes, normalizing terms to its root form. This process can greatly reduce vocabulary of the collection without much semantic loss. The stemmed terms are linked to its original form, which are preserved to be used in subsequent phases (i.e. labels generation).

Stop-words elimination

A stop-word itself doesn't bring any semantic meanings but in connection with other words can form meaningful phrases. Therefore, terms that occur in stop-word list are specially marked to be ignored from document index terms, but not removed (so it can uses in phrases extraction in label generation phase). Due to special nature of web document, some words like "web", "http", "site" appear very frequently, thus a stop-word list adapted to web vocabulary from [39] is used.

5.3.2. Document corpus building

As TRC utilizes Vector Space Model for creating document-term matrix representing documents.

Index terms selection

Index terms are selected from all unique stemmed terms after stop-words elimination and with regards to the following rules:

- digits and terms shorter than 2 characters are ignored
- terms contained in the query are ignored (as we are operating on the top search results for a query, terms from query will occurs in almost every snippet)
- Minimum Document Frequency filtering - term that occurs in less given threshold (e..g less than 2 snippets) are ignored as they will doubly contributes to document characterization

Selected index terms used to characterize documents are enumerated and a document-term frequency matrix is built. Let N be a number of document – search results snippets, and M is the number of selected index terms. The document-term frequency matrix is defined as follows:

$$TF = [tf_{i,j}]_{N \times M}$$

where $tf_{i,j}$ — number of occurrences of term j in document j. Each row $TF[i]$ of TF matrix is a characterization of the i-th document by means of term frequencies.

Term weighting

The TF*IDF term weighting scheme is applied for document-term matrix to create document-term weight matrix:

$$W = [w_{i,j}]_{N \times M}$$

where $w_{i,j}$ — weight of term j in i-th document. Each row $W[i]$ in the W matrix represents a characterization of the i-th document by means of weighted terms.

5.3.3. Tolerance class generation

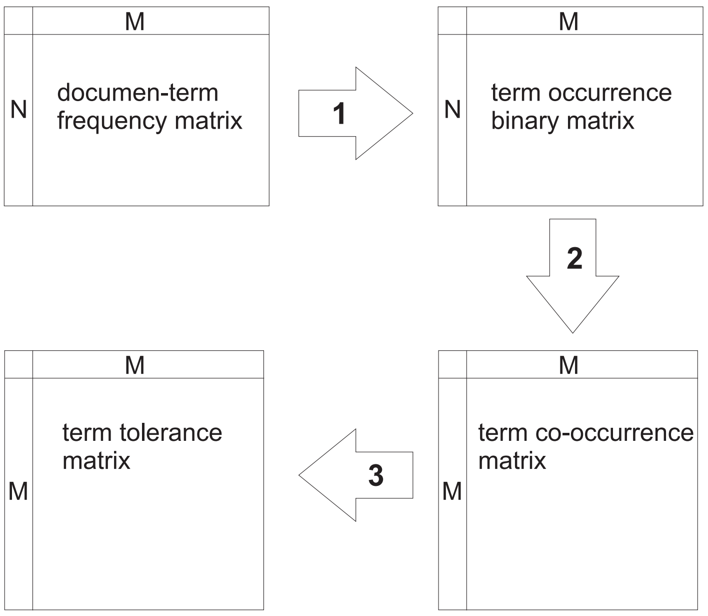

M
N
documen-term frequency matrix
1
M
N
term occurrence binary matrix
2
M
M
term tolerance matrix
3
M
M
term co-occurrence matrix

Figure 5.2: Process of generating tolerance classes

The goal of the tolerance class generation is to determine for each term, set of its related terms with regards to the tolerance relation – the tolerance class. This phase exists for computation optimization purpose, it ensures that the calculation of the upper approximation for a set of terms can be done quickly. Let define *term co-occurrence matrix* as

$$TC = [tc_{x,y}]_{M \times M}$$

where $tc_{x,y}$ is a *co-occurrence frequency* of two terms x, y — number of documents in the collection in which terms x and y both occur. Let tolerance relation R between terms be defined as

$$xRy \iff tc_{x,y} > \theta$$

where θ is called **co-occurrence threshold**. Having term co-occurrence matrix calculated we can define tolerance relation with different granularity by varying co-occurrence threshold.

The computation cost of step 1 is $O(N \times M)$, step 2 and 3 are both $O(M^2)$, altogether is $O(N \times M^2)$.

Algorithm 6 Tolerance class generation (Figure 5.2)

```
Input: $TF$ – document-term frequency matrix, $\theta$ – co-occurrence threshold
Output: $TOL$ – term tolerance binary matrix defining tolerance classes of term
1: Calculate a binary occurrence matrix $OC$ based on document-term frequency matrix $TF$
   as follows: $OC = [oc_{i,j}]_{N\times M}$ where
      $$oc_{i,j} = \begin{cases} 1 & \text{if } tf_{i,j} > 0 \\ 0 & \text{otherwise} \end{cases}$$
   Each column in $OC$ is a bit vector representing term occurrence pattern in a document
   — bit is set if term occurs in a document.
2: Construct term co-occurrence matrix $COC = [coc_{x,y}]_{M\times M}$ as follows: for each pair of
   term $x, y$ represented as pair of columns $OC[x], OC[y]$ – bit vectors – in the $OC$ matrix
      $$coc_{x,y} = \text{card}(OC^x \text{AND} OC^y)$$
   where AND is a binary AND between bit vectors and $card$ return cardinality – number
   of bits set – of a bit vector. $boc_{x,y}$ is the co-occurrence frequency of term $x$ and $y^1$.
3: given a co-occurrence threshold $\theta$, a term tolerance binary matrix $TOL = [tol_{x,y}]_{M\times M}]$
   can be easily constructed by filtering out cells with values smaller than threshold $\theta$ :
      $$tol_{x,y} = \begin{cases} 1 & \text{if } coc_{x,y} \geq \theta \\ 0 & \text{othewise} \end{cases}$$
   Each row in the resulting matrix forms a bit vector defining a tolerance class for given
   term: $tol_{x,y}$ is set if term $x$ and $y$ are in tolerance relation.
```

5.3.4. K-means clustering

TRC adapts a variation of K-means algorithm for creating groups of similar snippets. Several main steps of the algorithm are described below (see pseudo-code in Algorithm 7):

Initial cluster forming

Selected documents serve as initial cluster representatives $R_1, R_2, ..., R_K$.

Stop condition

The fact that the clustering algorithm is used as a post-retrieval[2] process puts a strict constraint on execution time, as users are not willing to wait more than a few seconds for a response. We decided to set a limit of **maximum number of iteration** for the K-means algorithm. Due to the nature of quick convergence of K-means algorithm, this limit allows us to reduce the response time of the clustering engine with insignificant lost in clustering quality.

Determining cluster representatives

Cluster representatives is determined as described in 5.1.1

[2]Results returned from search engine is processed on-the-fly by the clustering algorithm and presented to the user

Algorithm 7 Clustering phase of TRC algorithm

Input: D – set of N documents, K – number of clusters, δ – cluster similarity threshold
Output: K overlapping clusters of documents from D with associated membership value

```
1: Randomly select $K$ document from $D$ to serve as $K$ initial cluster $C_1, C_2, ..., C_K$.
2: repeat
3:   for each $d_i \in D$ do
4:     for each cluster $C_k$, $k = 1, .., K$ do
5:       calculate the similarity between document's upper approximation and the cluster
         representatives $S(\mathbf{U}_{\mathrm{R}}(d_i), R_k)$
6:       if $S(\mathbf{U}_{\mathrm{R}}(d_i), R_k) > \delta$ then
7:         assign $d_i$ to $C_k$ with the similarity taken as cluster membership: $m(d_i, C_k) =$
           $S(\mathbf{U}_{\mathrm{R}}(d_i), R_k)$
8:       end if
9:     end for
10:  end for
11:  for each cluster $C_k$ do
12:    determine_cluster_representatives($R_k$)
13:  end for
14: until stop_condition()
15: post-process unassigned documents
16: if necessary determine_cluster_representatives($R_k$) for changed clusters $C_k$
```

Nearest neighbor assignment

As a result of the restriction set by **cluster similarity threshold**, after all iterations there may exists document that has not been assigned to any cluster. In TRC, there are two possible options:

- create a special "Others" cluster with unassigned documents as proposed by [19]
- assigns these documents to their nearest cluster

For the later options, we have decided to assign document to its nearest neighbor's cluster.

Algorithm 8 Assignment of document to its nearest neighbor's cluster

```
for each unassigned document $d_u$ do
  find the nearest neighbor document $NN(d_u)$ with non-zero similarity
  amongst clusters which $NN(d_u)$ belongs to; choose the one $C_k$ that $NN(d_u)$ has strongest
  membership with
  assign $d_u$ to $C_k$ and calculate its cluster membership as $m(d_u, C_k) = m(NN(d_u), C_k) \cdot$
  $S(\mathbf{U}_{\mathrm{R}}(d_u), \mathbf{U}_{\mathrm{R}}(NN(d_u)))$
end for
```

5.3.5. Cluster label generation

As already pointed out, when evaluating clustering algorithms for search results, the quality of cluster label is as much important as the quality of the cluster itself. For labelling cluster we have decided to employs phrases because of its high descriptive power [44, 19]. We have adapted an algorithm for n-gram generation from [28] to extract phrases from contents of each cluster. Most descriptive phrases are chosen to serve as labels for cluster. Descriptiveness of phrases is evaluated by taking into consideration following criteria:

- frequency of the phrase in the whole collection
- frequency of the phrase inside a cluster
- length of the phrase, measured as number of words it is made from

Following the intuition of TD*IDF scheme, we hypothesize that phrases that are relatively infrequent in the whole collection but occurs frequently in clusters will be good candidate for cluster's label. We also prefer long phrases over shorter one.

5.4. Implementation

The TRC algorithm was implemented entirely in Java programming language, as a component within the *Carrot*^{2} framework. The choice of Java as an implementation language was motivated by many reasons. Java is a platform-independent, object-oriented language that encourage good software development practices by enforcing the usage of interfaces and proper organization of source code. There are numerous open-source libraries of software components necessary when developing clustering engines (e.g. XML processing, visualization, Natural Language Processing), available for free.

5.4.1. *Carrot*^{2} Framework

Carrot^2 is an open-source, data processing framework developed by Dawid Weiss [37]. It enables and ease experimentation of processing and visualization of search results. The *Carrot*^{2} architecture is based on a set of loosely-coupled but cooperating components that exchanges information with each other using XML messages. Standard components in the

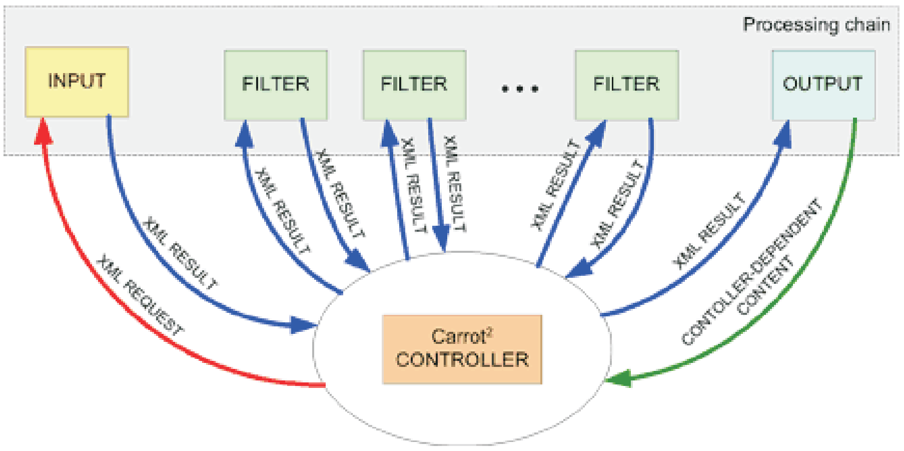


Figure 5.3: Data flow in *Carrot*^{2} framework

Carrot^2 framework are based on simple request-response paradigm – for a request sent to the component in a response is generated – supervised by special *controller* component. The data exchange between components is based on XML format and coordinated by the *Carrot*^{2} *Controller* (see 5.3). The controller serves also as an interface for external system as it accepts user's query request and returns processed, formatted results.

Components in the *Carrot*^{2} framework can be classified into three types:

input component serves as a data source. A typical input component could be a wrapper for search engine, that takes a user query, submit to the search engine and fetch the results returned for further processing.

filter component is used to transform one data set to another. An example of filter component could be stemming, stop-word removal or the whole clustering algorithm (as it is the case in this thesis).

output component's role is to turn incoming data to a presentable format for the users. For instance, output could transform XML data to HTML format so that it can be viewed in standard web browser.

5.4.2. TRC as a *Carrot*^{2} filter component

The TRC algorithm was implemented as a filter component within the *Carrot*^{2} framework. The use of *Carrot*^{2} framework instead of implementing the whole system by oneself was due to the following reasons:

- *Carrot*^{2} provide a powerful yet relatively simple framework designed specially with intention to serve as experimentation environment for clustering search results algorithms
- at the time writing, the *Carrot*^{2} framework has provided ready-to-use wrapper components as well as a visualization interface
- *Carrot*^{2} is tested and has proven itself to be reliable data processing framework as several clustering algorithms [39, 19, 40] has been implemented with success

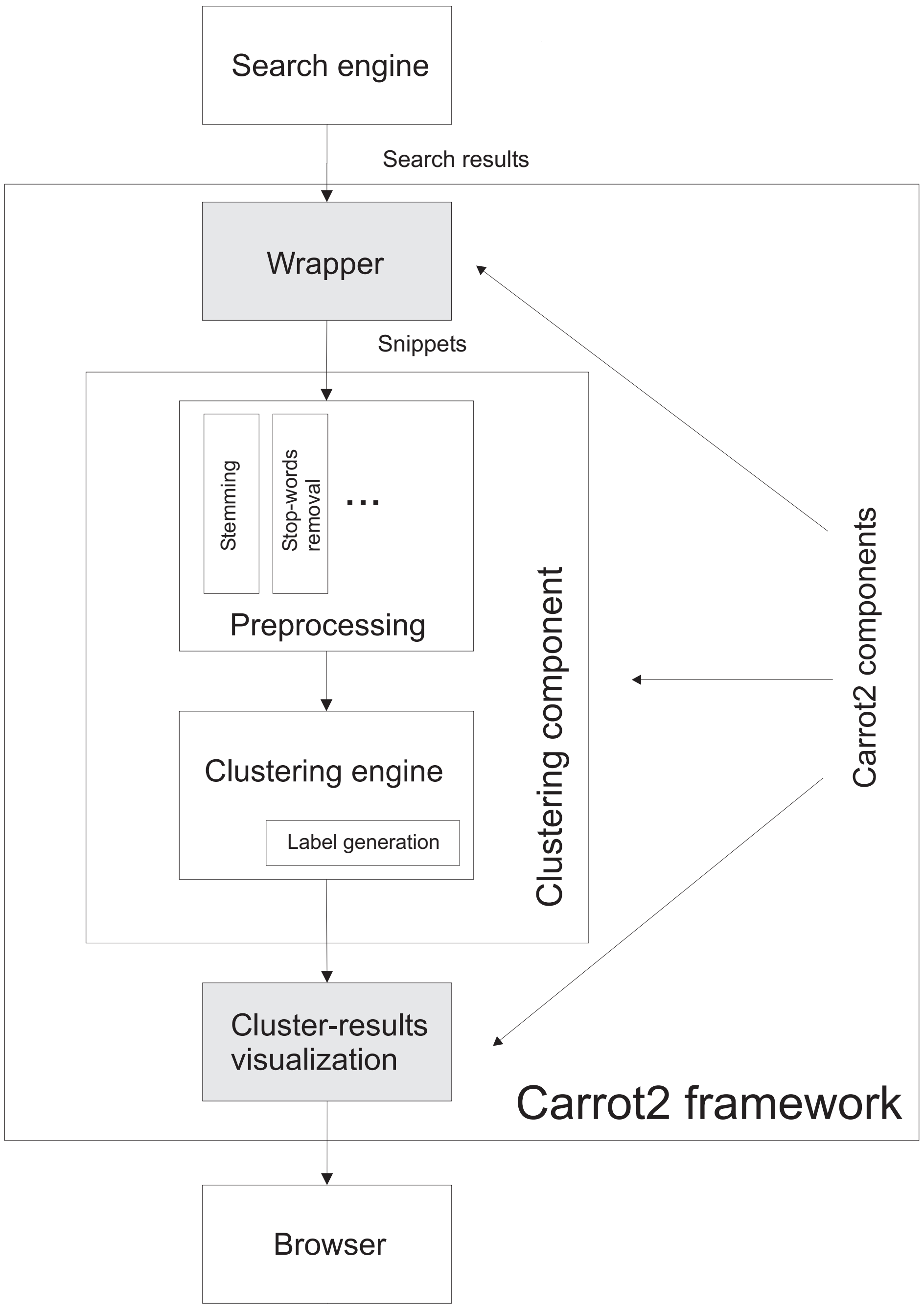


Figure 5.4: TRC as a filtering components within *Carrot*^{2} framework.

Chapter 6

Evaluation, experiments, comparisons

This chapter presents an evaluation of our proposed TRC algorithm. We start by describing current methods for evaluation document clustering, continuing with detailed description of evaluation approach that we have adapted and its outcomes. Our evaluation concentrates on following goals:

1. investigating if the document's enrichment offered by tolerance space can improve cluster's quality
2. compare clustering quality of TRC with other search results clustering algorithm

6.1. Current evaluation methods

When looking at current approaches to evaluation for document clustering, one can observes two categories of methods.

6.1.1. Ground truth

First category of evaluation methods is originated from Information Retrieval domain and is based on a *ground truth set*[30] — a reference set of documents with accompanying information about relevance (documents are pre-classified as relevant or not, or assigned a relevance degree) or partition (documents are assigned to pre-identified groups). The tested algorithm is executed with those input documents and the results compared with original classification or partition. Methods of evaluation then vary on how to prepare the test collection and what metrics are used to measure the quality of results.

Precision-Recall

Precision and recall are standard measures used for evaluating quality of information retrieval systems. Let D be a set of documents, $Q \subseteq D$ set of all relevant documents, A set of documents retrieved by system and $A_Q = A \cap Q$ set of relevant document retrieved by system. The precision p and recall r can be defined as follows [25]:

$$p = \frac{|A_Q|}{|A|}$$

$$r = \frac{|A_Q|}{|Q|}$$

An ideal solution would be a system producing results with both precision and recall equals 1, but in practice algorithms are trying first to achieve as much precision as possible while maintaining reasonable recall.
With the assumption that resulting cluster is ordered in some way (e.g. in [11] users are assumed to be able to locate the "best" clusters), these measures can be adapted for evaluation clustering search results [44].

Merge-then-cluster

For document clustering, a useful evaluation technique is to take documents from different sources/domain (so that it is inherently about different topics) and merge them together to create a test collection. The performance of clustering algorithm is then measured based on comparison of the resulting clusters and original classes of documents. A common metric used is the entropy measure, which considers the distribution of various classes of documents amongst clusters. Entropy for a clustering solution, as in [47] is defined as:

$$E = \sum_{r=1}^{k} \frac{n_r}{n} E(C_r)$$

where

$$E(C_r) = -\frac{1}{\log q} \sum_{i=1}^{q} \frac{n_r^i}{n_r} \log \frac{n_r^i}{n_r}$$

and C_r is a cluster with size n_r, q – number of document classes, n_r^i – number of documents of i-th class that was assigned to cluster C_r.
Methods of this category depend largely on quality and availability of test collection. For example, document collections with pre-classification performed by expert may be debatable as there is always some subjectivity involves, while clustering document from different sources could reveal the topic sharing between document originally not related. The main advantage of ground-truth methods is that once a test collection is available, it can be automatically performed, thus the algorithm can be easily tested with various parameters and/or compared with other algorithms.

6.1.2. User feedback

As many applications of document clustering were developed with purpose to help users in various information task, user evaluation seems to be natural way to assess its usefulness. User evaluation methods can be based on direct reports from the users (e.g. survey) or on automatic analysis of user interaction with the system (e.g. log analysis, web usage mining). An extensive example of user evaluation is presented in [44] and minor attempts were done in [19][40]. The problem with user evaluation is that for providing statistically significant results it must be done on large scale. Collecting surveys from a large number of users or attracting users to use an experimental system for a reasonable period of time (so that extensive log could be gathered) is a non-trivial task.

6.2. Our evaluation

For limited time and resources assigned for this thesis, we could not perform a significant user evaluation. Also the lack of standard collection of web snippets has prevented us from doing extensive evaluation. Instead, we have carried only an expert (by ourselves) evaluation of the proposed algorithm in a small own-authored set of test data and compared our solution with other approaches based on several examples.

6.3. Test data collection

Due to aforementioned lacks of standard collection for testing web search results clustering, we had to build a small test collection. For this purpose, we have defined a set of queries for which results were collected from major search engine Google [10] to form test data collection.

6.3.1. Test queries

The test set is composed from queries representing subjects of various degree of specificity, in order to test algorithm behavior on data with different vocabulary characteristic. The first three queries represent very general concepts and are frequently used as a test by authors [44, 39, 19, 40] of search results clustering algorithm. Next three queries are more specific subjects but broad enough to have interesting subclasses. Last three queries are about relatively specific topics and were chosen to test algorithm on highly cohesive vocabulary. In Table 6.1, figures in the second column are the *approximated number of results* for a queries retrieved from Google. Looking at these figures gives us some additional intuition about generality (or specificity) of concepts represented by query, or just popularity of each subject on the Web.

Query	**Results count**[1]	**Specificity**[2]
java	23400000	low
clinton	4270000	low
jaguar	2580000	low
"data mining"	1080000	medium
wifi	864000	medium
clustering	810000	medium
voip	916000	high
"rough sets"	748	high
"search results clustering"	105	high

Table 6.1: Queries used to generate test collection.

6.3.2. Data characteristic

Table 6.2 shows some statistics of snippets collection retrieved from Google search engine for set of test queries. Both stemming and stop-word list were used to filter out unnecessary terms. Additionally Minimum Document Frequency filter was used to remove terms that occurs in less than 2 documents. Thus, the indexing vocabulary could be reduced by about 70% (see Table 6.3). On average a snippets is indexed by 6 to 10 terms, which is about 2-3% of total vocabulary. This is very specific for search results snippets and in consequences the representing vectors are very sparse. Worth noticed in the results of term filtering, some document may be left represented by none terms. In our test collection, this happens in case of several queries (see last column of Table 6.2). This is because all terms in document were too specific and existed only in this document. Therefore, they were removed by Minimum Document Frequency filter. We can view these documents as an outlier area.

Query	Snippets	Terms	Terms per snippet Average	Min	Max	Zero-terms snippets
java	200	332	7.28	0	15	1
clinton	200	337	7.305	0	13	3
jaguar	200	339	6.595	0	13	3
"data mining"	200	319	8.815	0	16	2
wifi	200	340	6.915	1	19	0
clustering	195	319	7.456	0	16	2
voip	200	355	8.235	0	17	5
"rough sets"	146	244	9.089	1	18	0
"search results clustering"	68	149	12.279	2	19	0

Table 6.2: Characteristic of snippets retrieved from Google for defined queries.

6.3.3. Effects of terms filtering

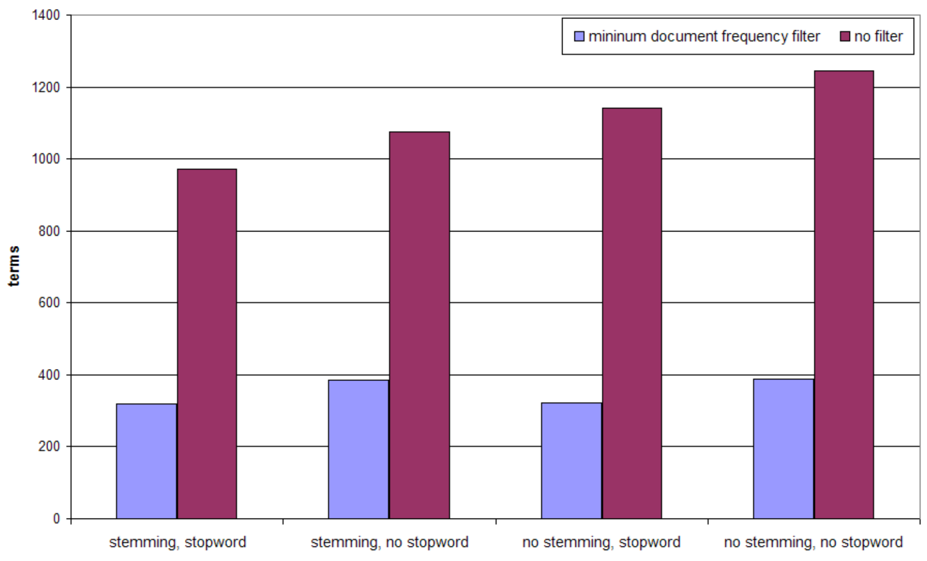


Figure 6.1: Effects of terms filtering on number of terms (data from Table 6.3)

The effects of different filtering techniques on vocabulary (i.e. number of unique terms) are showed in an example (see Table 6.3) of 200 snippets returned by search engine for query "data mining". Figures from the last column of Table 6.3 shows the percentage of vocabulary terms that has been reduced by the usage of both stemming and stop-word list (numbers in 1-st data column) in comparison with a situation when no stemming and stop-word list were used (figures from 4-th data column). The usage of stemming and stop-word list results in about 20% reduction of vocabulary (see the last column of Table 6.3). More notably however, is fact that the simple Minimum Document Frequency (MDF) filter which removes terms occurring in less than given threshold of documents (2 in this case) could reduce vocabulary by nearly 70% (compare the 1-st data row with the 2-nd data row in Table 6.3), thus greatly improves computation performance. When looking at data in the Table 6.3, an interesting observation can be made. In case when no additional filter is used, with changing combination of stemming/stop-word, there is a consecutive increase in number of terms (Table 6.3, data in the second row, moving from left to right). When MDF filter is used (Table 6.3, first row), in contrast, there is a strong decline in number of terms when moving from combination stemming/no stop-word to no stemming/stop-word. We will explain this on the following example

Example 6 *Let consider following words: "computer", "computers", "computing", "compute" and let assume that each occurs in exactly one document in the collection. The application of stemming brings all these words into a common form "comput" which will have a document frequency of 4 and will remain untouched after MDF filter. When no stemming is applied, document frequency of each of these words are 1 and they may be removed by MDF filter with threshold 2 (only terms occurring in 2 or more documents are preserved).*

	stemming stopword	**stemming no stopword**	**no stemming stopword**	**no stemming no stopword**	**% reduced**
MDF filter	319	386	323	389	17,99%
no filter	971	1074	1142	1246	22,07%
% reduced	67,14%	64,05%	71,71%	68,78%	**74,39%**[3]

Table 6.3: Number of unique terms for 200 snippets retrieved for query "data mining" and the effects of stemming, stop-words and MDF filtering on collection's vocabulary.

6.4. Tolerance classes generation

For testing the effects of varying *co-occurrence threshold* on generation of tolerance classes we have defined the two measures: *number of tolerance classes* and *tolerance richness.* For precise definition of these measures, we will introduce the following metaphor:

- lets each index term t_i is treated as a node v_i in a graph G
- edges in graph G represent the tolerance relation between terms: if terms t_{i1}, t_{i2} are in tolerance relation then its equivalent nodes v_{i1}, v_{i2} in graph G are connected with edge $e_{i1,i2}$
- self-edges – edge connecting nodes with its self (this can happen as with 4.3.1 each terms is tolerance relation with itself) – are ignored

number of tolerance classes is a number of terms with non-trivial tolerance class – classes with size ≥ 2 (i.e. number of nodes that have edges in graph G)

tolerance richness is a number of edges in graph G. This measures the richness of the tolerance relation and also reflects the total size of all tolerance classes.

6.4.1. Size and richness

Tolerance class generation was carried on sets of 200 snippets for test queries (see Table 6.2). The number of tolerance classes generated stabilizes pretty quickly (see Figure 6.2, 6.3) with the increase of *co-occurrence threshold.* The similarity can be observed for tolerance richness (see Figure 6.4, 6.5), also starting to stabilize after certain threshold (ranging in 4-8, varies from query). This confirms believes that the co-occurrence relation represented by tolerance classes of term is backed by existing underlying relation between terms. Some backing examples of generated tolerance classes are in Table 6.4 and 6.5.

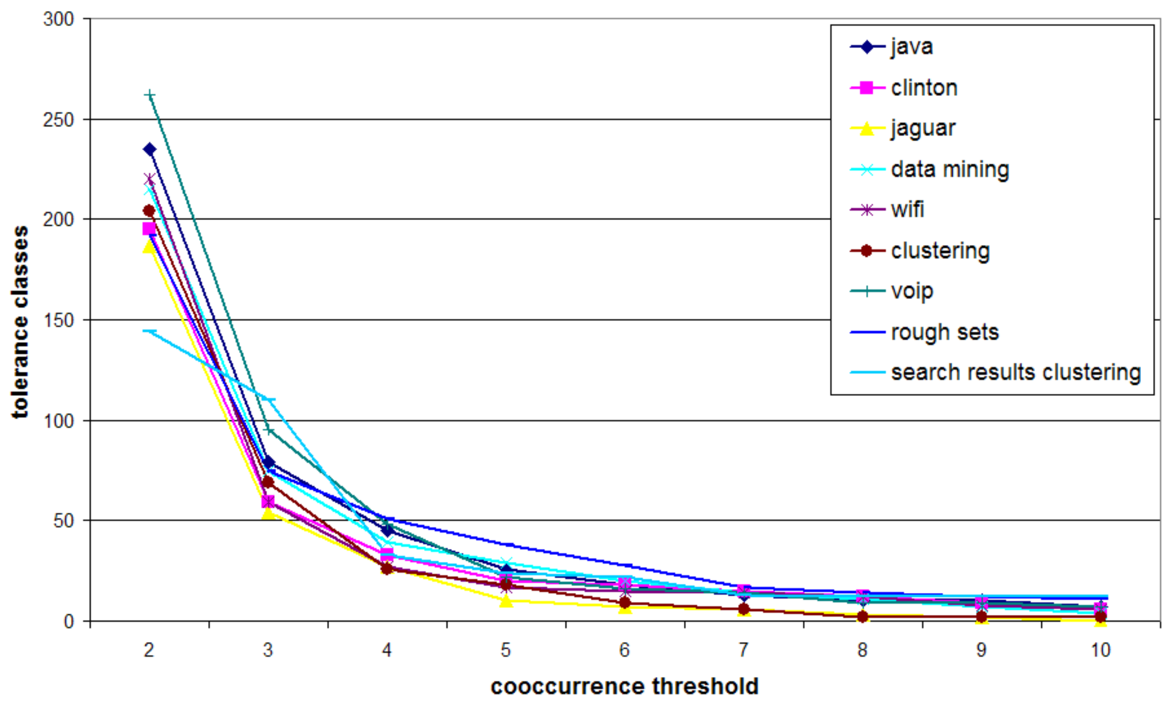


Figure 6.2: Number of tolerance classes with various co-occurrence thresholds

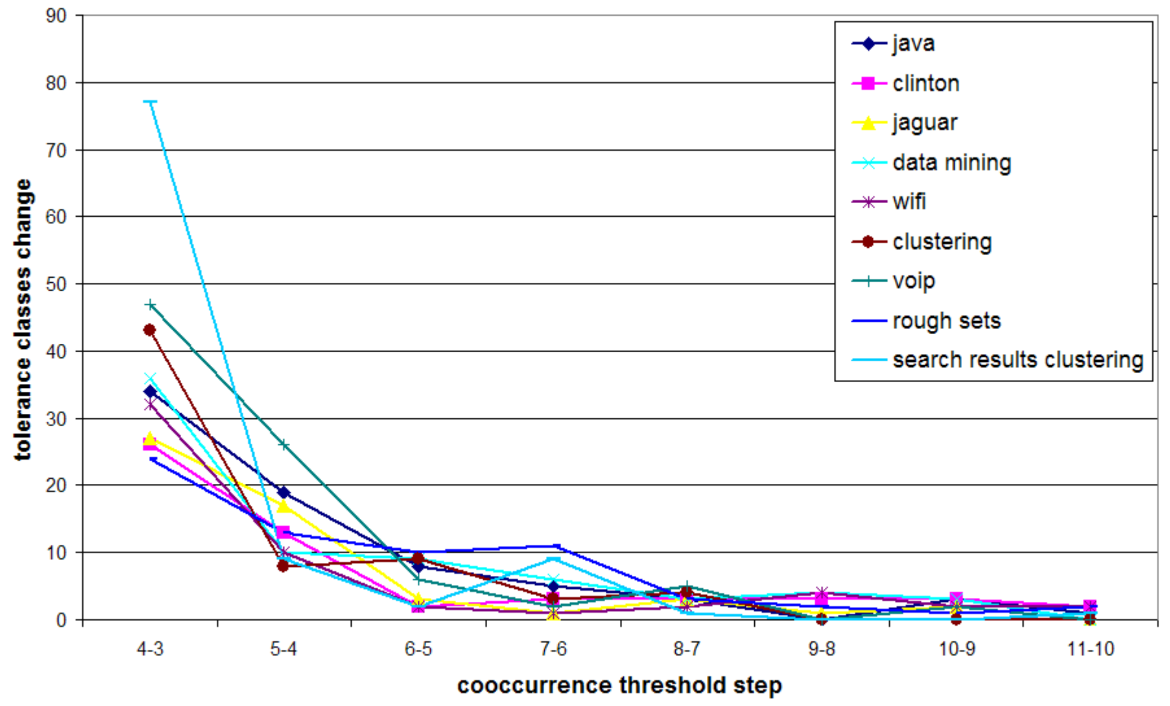


Figure 6.3: The change in number of tolerance classes while varying co-occurrence threshold

6.5. Upper approximation enrichment

In this section, we will try to answer the questions whether the usage of upper approximation can enrich document representation, and what effect will it have on inter-document

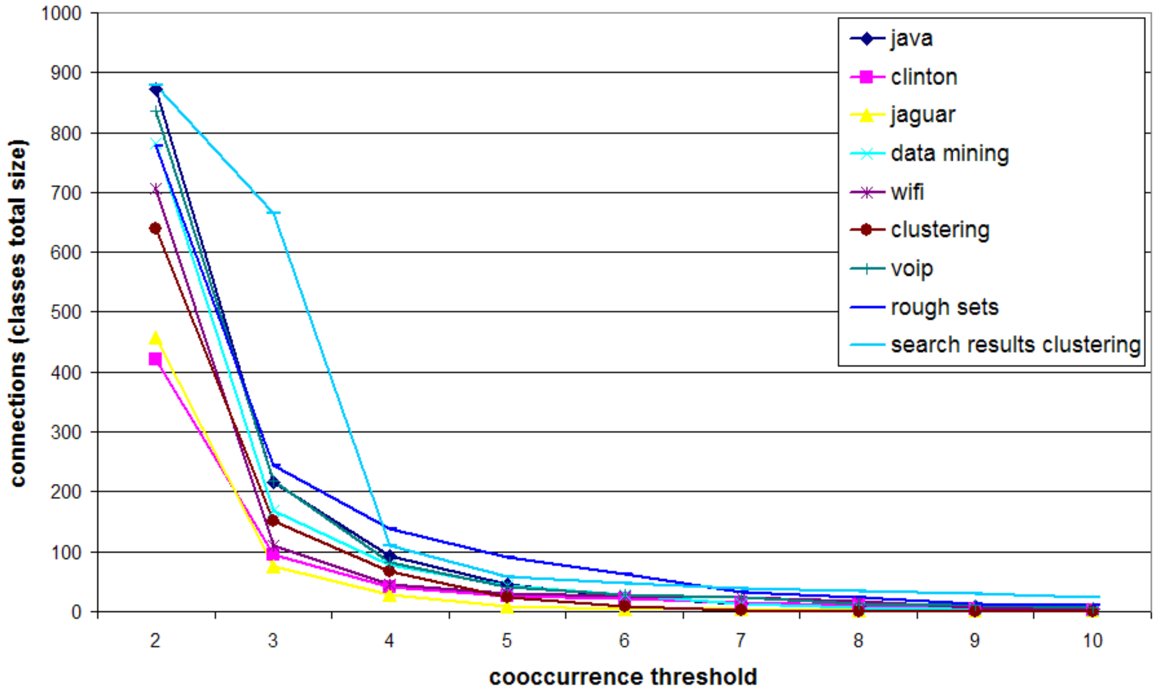


Figure 6.4: Number of connections between terms in tolerance classes (total size of all tolerance classes)

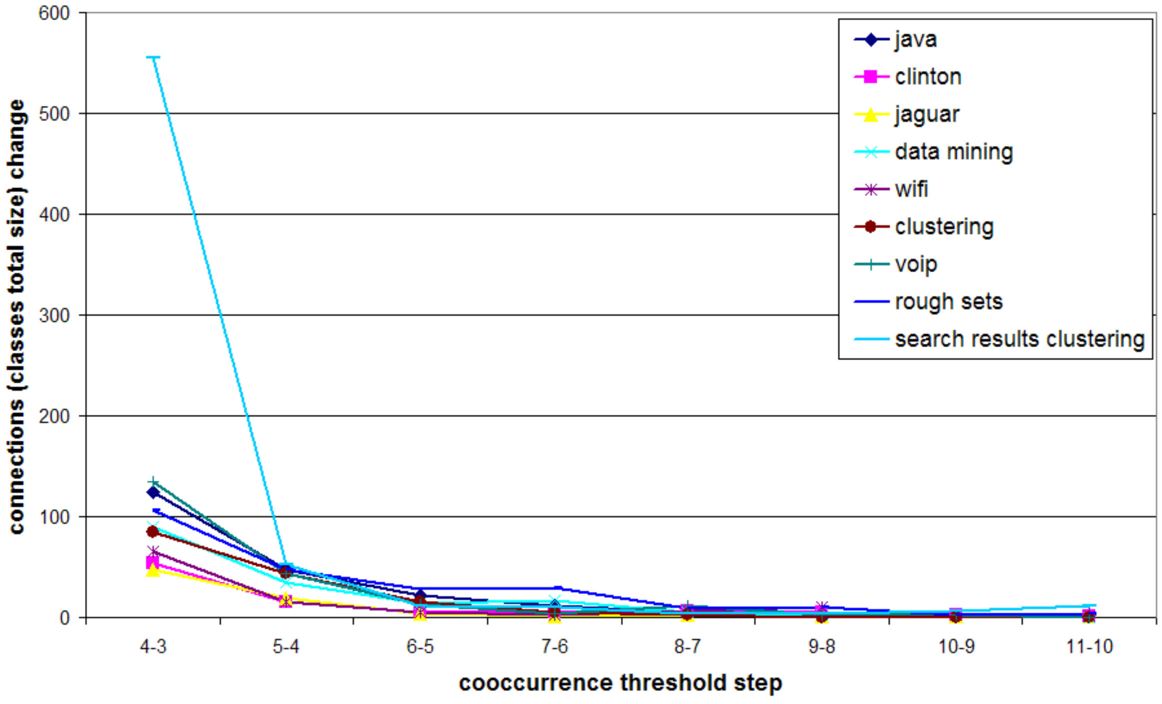


Figure 6.5: The change in number of connections between terms in tolerance classes (change in total size of all tolerance classes) while varying cooccurrence threshold

Term	Tolerance class
Intelligence	Intelligence, Business
Discovery	Discovery, SIGKDD, KDD, Knowledge, International, Conference, ACM
SIGKDD	Discovery, SIGKDD, Knowledge, ACM
Machine	Machine, Learning
studies	studies, White, case, Paper, webcasts
White	studies, White, case, Paper, webcasts
case	studies, White, case, Paper, webcasts
Paper	studies, White, case, Paper, webcasts
services	services, Software
statistical	statistical, Learning
Learning	Machine, statistical, Learning
Business	Intelligence, Business
KDD	Discovery, KDD, Knowledge, International
Software	services, Software
Knowledge	Discovery, SIGKDD, KDD, Knowledge, International, Conference, ACM
International	Discovery, KDD, Knowledge, International, Conference, Workshop
webcasts	studies, White, case, Paper, webcasts
Conference	Discovery, Knowledge, International, Conference
Workshop	International, Workshop
ACM	Discovery, SIGKDD, Knowledge, ACM

Table 6.4: Tolerance classes generated from snippets of query "data mining" with co-occurrence threshold set to 6.

Term	Tolerance class
Computing	Computing, data, Current, Trends, Granular, International, mining, Conferences, RSCTC
Discovery	Discovery, Polkowski, knowledge, Lech, applications
Polkowski	Discovery, Polkowski, Skowron, knowledge, Lech, applications
data	Computing, data, Granular, mining
Skowron	Polkowski, Skowron, Andrzej
Current	Computing, Current, Trends, International, Conferences, RSCTC
knowledge	Discovery, Polkowski, knowledge, Lech, applications
Trends	Computing, Current, Trends, International, Conferences, RSCTC
Granular	Computing, data, Granular, mining
Andrzej	Skowron, Andrzej
International	Computing, Current, Trends, International, Conferences, Workshop
mining	Computing, data, Granular, mining
Conferences	Computing, Current, Trends, International, Conferences
RSCTC	Computing, Current, Trends, RSCTC
Workshop	International, Workshop
Lech	Discovery, Polkowski, knowledge, Lech, applications
applications	Discovery, Polkowski, knowledge, Lech, applications

Table 6.5: Tolerance classes generated from snippets of query "rough sets" with co-occurrence threshold set to 7.

similarities. Most concepts like document representation, tolerance classes, inter-document similarities, are represented in our system as a matrix. Therefore, we want to examine the matrix of underlying concepts with and without upper approximation in use. Generally, we are interested in the some kind of "richness" of these matrices. For evaluating matrix's richness, let introduce *the cardinality* measure for matrix:

$\text{card}_{threshold}$ – number of matrix's entries, that values are greater than given threshold (for *threshold* $= 0$ it is equivalent to number of non-zero entries).

For each concept that we want to investigate, it equivalent matrix is constructed, firstly without using upper approximation and later utilizing upper approximation with various co-occurrence thresholds. The above defined cardinality is calculated for each of the constructed matrix and compared.

Let M denotes a matrix representing investigated concept without usage of upper approximation and M_U denotes the matrix of the same concept but utilizing upper approximation. The cardinality with specified threshold t for these matrices is denoted $\text{card}_t(M)$, $\text{card}_t(M_U)$ adequately. The relative improvement of matrix's richness brought in by upper approximation can be measured as follows:

$$\text{relative richness improvement} = \frac{\text{card}_t(M_U) - \text{card}_t(M)}{\text{card}_t(M)}$$

6.5.1. Upper approximation of document representation

We want to measure how much "richer" is the document representation when upper approximation is used, compared to the original representation. Recall from Section 3.2 that documents are represented in vectors, together composing a document-term matrix.

The richness of document-term matrix can be measured as number of non-zero entries (card_t with $t = 0$). The cardinality for the document-term matrix without and with upper approximation was calculated using various co-occurrence thresholds and presented in figure 6.6. The improvement shown in figure 6.6 is very similar in nature to that of tolerance classes (see Figure 6.2), It is characterized with strong decrease at the beginning and quick stabilization starting from co-occurrence threshold of 5. Figure 6.7 shows relative improvement of document-term matrix's richness when upper approximation is applied with co-occurrence threshold of 5. This is a clear proof that upper approximation can significantly improve the richness of document representation for all tested queries.

6.5.2. Inter-document similarity

Similarity between two documents — inter-document similarity — calculated using one of defined measure (see 3.2.3) quantifies the relatedness between documents. The main purpose upper approximation is to enhance the association between documents (i.e. increase the similarity between related documents). To measure that enhancement, we have to examine inter-document similarity for all pairs of documents in the collection. Inter-document similarity for the whole document collections can be represented in a squared, symmetrical matrix called inter-document similarity matrix. The inter-document similarity matrix can be defined as:

$$IDS = [s_{i,j}]_{N \times N}$$

where $s_{i,j}$ is the similarity between document i and j in the collections of N documents. $s_{i,j}$ usually has value between 0 and 1 depending on used similarity measure. In experiments below, the cosine coefficient (see Section 3.2.3) was used. Because the inter-document similarity matrix is symmetrical and the entries in the diagonal (self similarities) are interested, in calculation of matrix's cardinality only values in the upper diagonal were considered.

We calculated inter-document similarity matrix for all snippets collection (as defined in 6.2) with standard document representation and the one using upper approximation. The cardinality for different threshold t ($t = 0, 0.33, 0.5, 0.67$) measured for these matrices is presented in Figures 6.8-6.11.

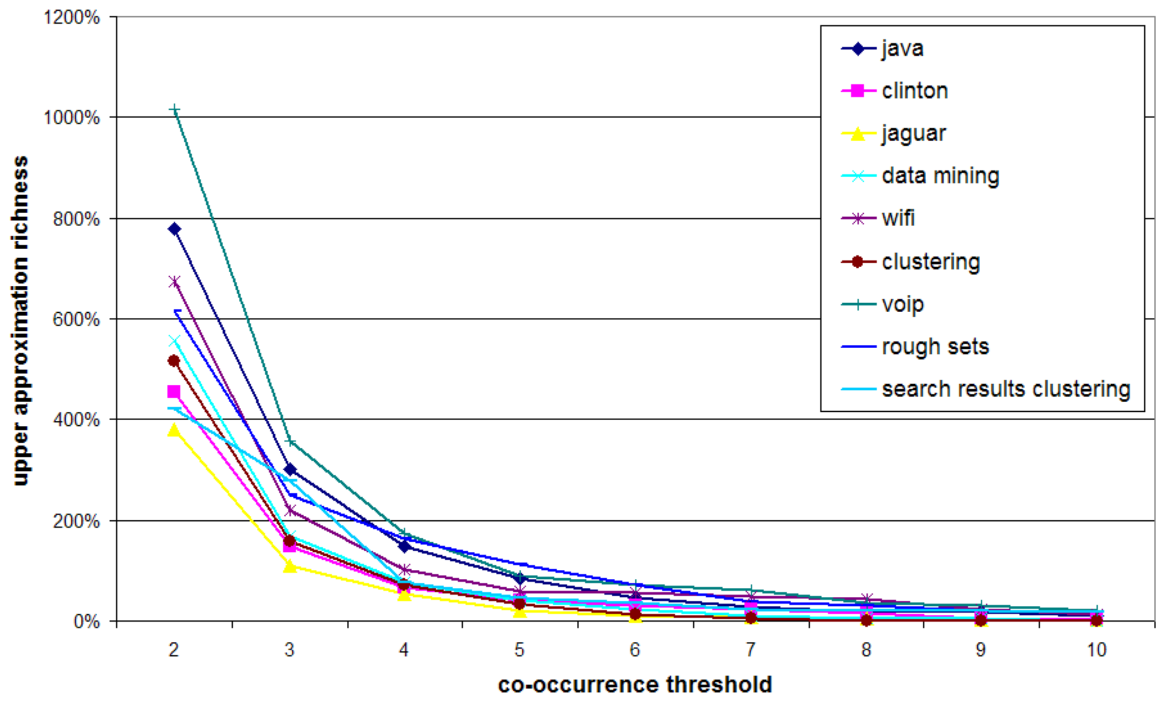


Figure 6.6: Relative improvement of document-term matrix when using upper approximation.

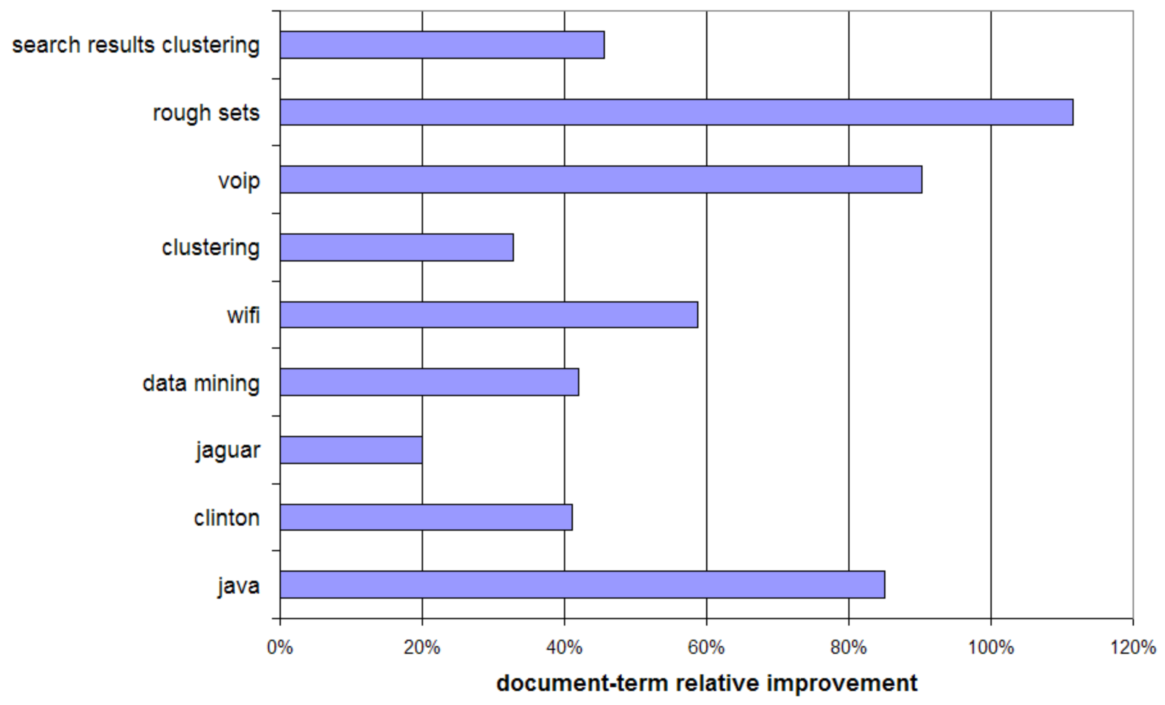


Figure 6.7: Relative improvement of document-term matrix when using upper approximation with co-occurrence threshold = 5.

Figures 6.8 - 6.11 shows plot of relative improvement of inter-document similarity for similarity threshold $0, 0.33, 0.5, 0.67$ with increasing co-occurrence threshold. From these

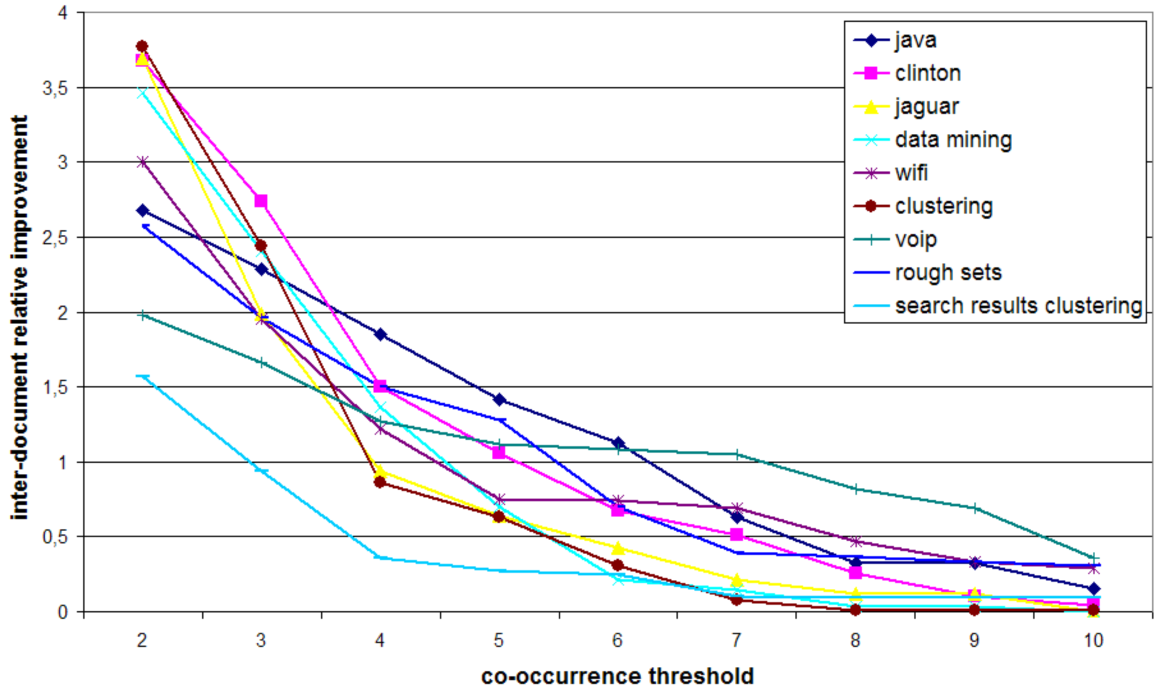


Figure 6.8: Relative improvement of inter-document similarity matrix measure for $t = 0$

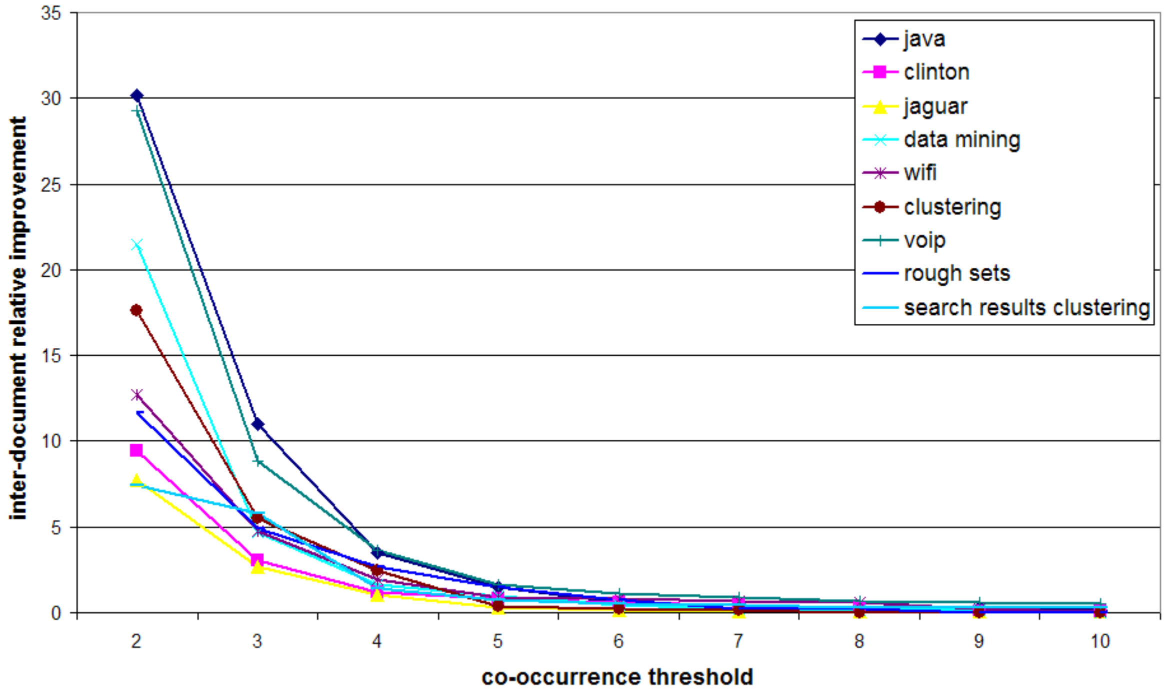


Figure 6.9: Relative improvement of inter-document similarity matrix measure for $t = 0.33$

figures, it can be observed that the enrichment of upper approximation had indeed improved inter-document similarities for all queries. The level of improvement varies with different queries depending on the co-occurrence threshold. Some queries like ”java”, ”clinton”, ”voip”,

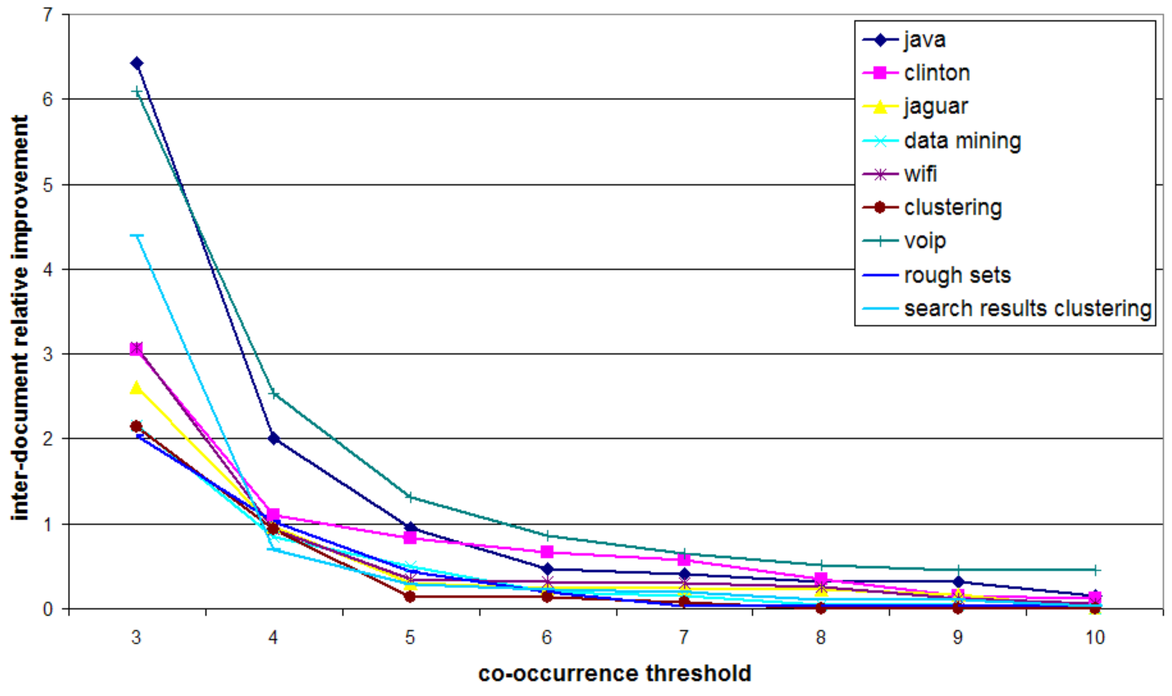


Figure 6.10: Relative improvement of inter-document similarity matrix measure for $t = 0.5$

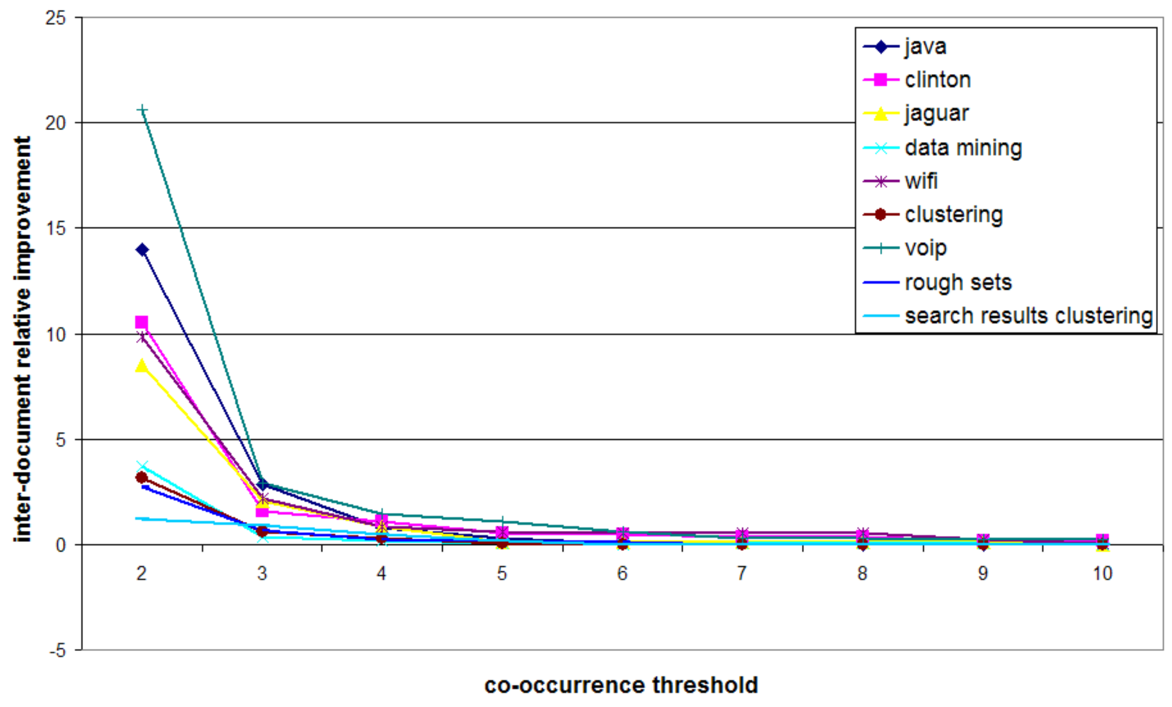


Figure 6.11: Relative improvement of inter-document similarity matrix measure for $t = 0.67$

achieved significantly better level of improvement than others ("jaguar", "clustering"). This is showed in figure 6.12 where we set co-occurrence threshold at 5 and plot relative improvement for various thresholds t. It is promising that improvement has happened in all range of values.

Not only low-range values (e.g. increased number of non-zero entries) but also higher value similarities (with value over 0.67) were significantly improved. This is very important for clustering quality as this could potentially pulled related documents together (and thus forms better clusters).

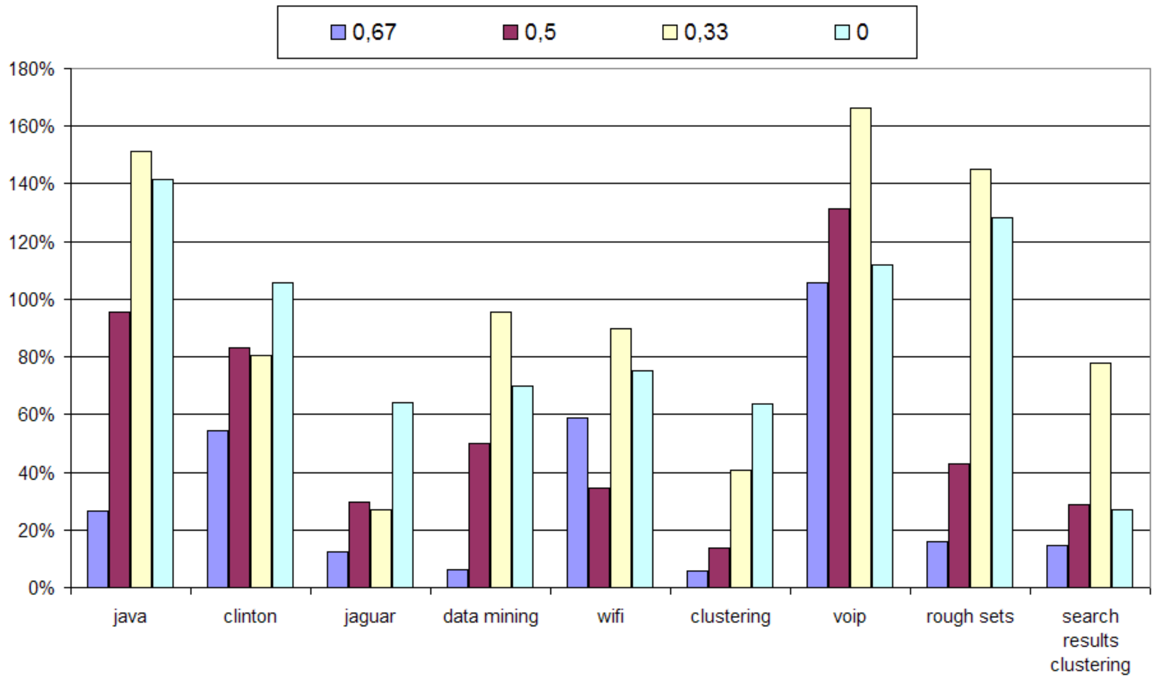


Figure 6.12: Relative improvement of inter-document similarity matrix measure with co-occurrence threshold = 5 and various $t = 0, 0.33, 0.5, 0.67$

The following experiment analyzes the effects of upper approximations in more details, examining the improvement in different range values of the inter-document similarity matrix. The range of similarity values was divided into 10 sub-segments $[0, 0.1], (0.1, 0.2], (0.2, 0.3], ..., (0.9, 1]$ (as cosine coefficient returns values between 0 and 1 inclusive). Entries of inter-document similarity matrix were adequately divided into 10 groups regarding their values. The effects of upper approximation for different values range is presented in the following example of snippets collection for query "clinton".

In table 6.6, the first section shows total number of entries in the original inter-document similarities and its distribution across value ranges followed by percentages that each group contributes to the total number of entries. The first observation is that most entries (nearly 90%) fell in the $[0, 0.1]$ range. Secondly, one can spots amongst the rest entries (with values over 0.1) that there are three types of group (we will name it as "low-range", "mid-range", "high-range") by looking at number of entries in each group. The first type is a group of entries in range $(0.1, 0.4]$, each group with over a hundred entries but less than a thousand, together contributing to about 5% of all entries. Next are groups of entries within range $(0.4, 0.8]$, each with less than 100 entries, together consisting less than 1% entries. The third type is about 5% of all entries forming a group similarity over 0.8.

The second and third section of the table show number of entries in groups when upper approximation were used with different threshold and the relative changes in comparison with original matrix. We noted that upper approximation with small co-occurrence threshold (< 5) can greatly improve "low-range" similarity entries, with less effects on "mid-range" and leaving "high-range" entries practically unchanged (only entries from $(0.8, 0.9]$ "moving" up to $(0.9, 1]$). With threshold over 5, "low-range" entries are consistently improved. The "mid-range" is starting to converge to the original number of entries, except entries in range $(0.4, 0.6]$ which are slightly improved. "High-range" entries remain unchanged overall.

Finally, the third section ("Aggregated sum of changes"), shows how much changes were introduced by upper approximation for group of entries within given range up to 1. For example, take a look at the 5-th data column (entries within range $(0.4, 0.5]$) in third section. With threshold at 5 we have value 102, which means that there are that many more entries in range $(0.4, 1]$ in compared with original similarity matrix. The 1-st column of this section contains all zeroes as the overall number of entries (in range $[0, 1]$) is unchanged.

The third section was meant to show how much improvement can be achieved at different range. For example, for purpose of clustering, we could set a similarity threshold of 0.5, so that documents with similarity greater than that threshold would be considered similar or belongs to one group/cluster. Looking at the third section at appropriate column, we could see how many documents can be "pulled" or grouped together if upper approximation is used with given co-occurrence threshold.

Range	[0,0.1]	(0.1,0.2]	(0.2,0.3]	(0.3,0.4]	(0.4,0.5]	(0.5,0.6]	(0.6,0.7]	(0.7,0.8]	(0.8,0.9]	(0.9,1]
					Original					
19900	**17702**	**931**	**382**	**172**	**72**	**25**	**13**	**8**	**595**	**0**
%	0.8895	0.0467	0.0191	0.0086	0.0036	0.0012	0.0006	0.0004	0.0298	0
Threshold					**Upper approximation**					
2	10343	4147	2083	1120	820	488	220	67	13	599
3	13814	2974	1375	628	325	123	44	19	3	595
4	15724	2125	840	347	171	58	25	14	1	595
5	16343	1764	693	285	135	50	22	12	1	595
6	16697	1576	575	271	109	48	18	10	1	595
7	16880	1457	545	261	89	45	17	11	595	0
8	17222	1226	498	211	86	38	14	10	595	0
9	17539	1044	409	181	79	32	13	8	595	0
10	17620	983	399	175	76	31	13	8	595	0
					Changes					
2	-7359	3216	1701	948	748	463	207	59	-582	599
3	-3888	2043	993	456	253	98	31	11	-592	595
4	-1978	1194	458	175	99	33	12	6	-594	595
5	-1359	833	311	113	63	25	9	4	-594	595
6	-1005	645	193	99	37	23	5	2	-594	595
7	-822	526	163	89	17	20	4	3	0	0
8	-480	295	116	39	14	13	1	2	0	0
9	-163	113	27	9	7	7	0	0	0	0
10	-82	52	17	3	4	6	0	0	0	0
					Aggregated sum of changes					
2	0	7359	4143	2442	1494	746	283	76	17	599
3	0	3888	1845	852	396	143	45	14	3	595
4	0	1978	784	326	151	52	19	7	1	595
5	0	1359	526	215	102	39	14	5	1	595
6	0	1005	360	167	68	31	8	3	1	595
7	0	822	296	133	44	27	7	3	0	0
8	0	480	185	69	30	16	3	2	0	0
9	0	163	50	23	14	7	0	0	0	0
10	0	82	30	13	10	6	0	0	0	0

Table 6.6: Characteristic of inter-document similarity matrix for query "jaguar".

6.6. Example of results

This section aims to provide an expert evaluation of TRC algorithm based on real-word examples. We will take the role of the expert and discuss the results of clustering performed by TRC on number of test queries (see Table 6.1). We will examine the effects that different algorithm parameters have on the quality of clustering. Finally we compare results produced by TRC with some other search results clustering approaches.

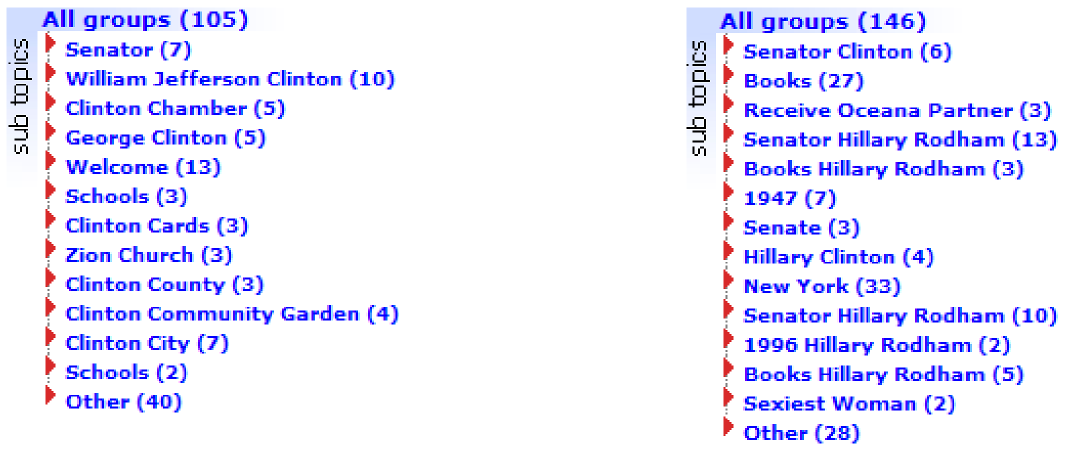
All experiments were carried out on search results returned from Google search engine [10]. Most examples are using the following parameters set, if not stated otherwise specifically.

Parameter	Value
Search engine	Google
Number of snippets	100
Initial clusters	30
Co-occurence threshold	5
Similarity threshold	0.25

Table 6.7: Default parameters for experiments

6.6.1. Topic generality

This experiment was performed to investigate the influence of topic generality on generated clusters. To continue the tradition in search results clustering evaluation (see [44, 39, 19, 40]), we will use the pair of queries: ”clinton” (as a broad topic example) and ”hillary rodham clinton” (as a more specific topic).



All groups (105)
- Senator (7)
- William Jefferson Clinton (10)
- Clinton Chamber (5)
- George Clinton (5)
- Welcome (13)
- Schools (3)
- Clinton Cards (3)
- Zion Church (3)
- Clinton County (3)
- Clinton Community Garden (4)
- Clinton City (7)
- Schools (2)
- Other (40)

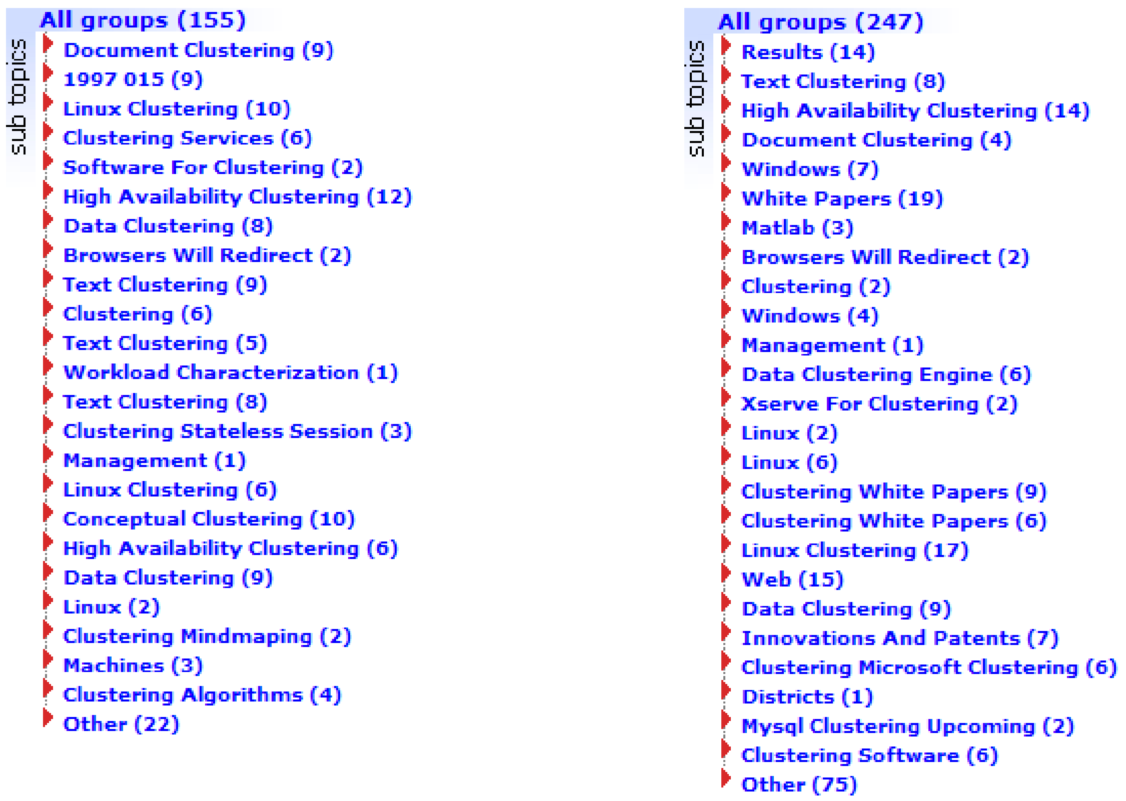
All groups (146)
- Senator Clinton (6)
- Books (27)
- Receive Oceana Partner (3)
- Senator Hillary Rodham (13)
- Books Hillary Rodham (3)
- 1947 (7)
- Senate (3)
- Hillary Clinton (4)
- New York (33)
- Senator Hillary Rodham (10)
- 1996 Hillary Rodham (2)
- Books Hillary Rodham (5)
- Sexiest Woman (2)
- Other (28)

Figure 6.13: Example of clustering results produced by TRC for query ”clinton” (left) and ”hillary rodham clinton” (right).

TRC performs reasonably well on this task, but it is noticeable that the ”Others” groups is somewhat large. It seems that TRC performs better for queries with precise and distinct topics, while struggle with more mixed-topics documents.

6.6.2. Data size

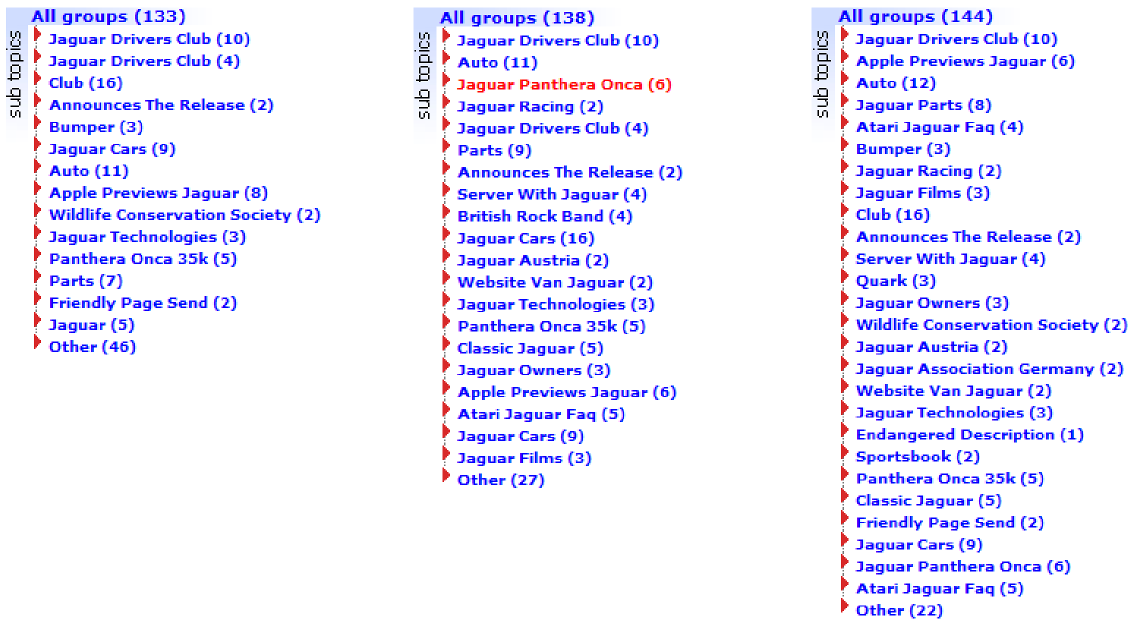
The increase of number of snippets has effects not only on computation time but also on quality of produced clusters. Visible is the problem of initial clusters (see 6.14). It is by default set to 30 which seems work reasonably good for 100 snippets but too small for 200 snippets. This resulted in a lot of document in the ”Others” group, while at the same time there are several groups with similar labels. The labelling algorithm also shows its weaknesses as it’s only able to look locally in the cluster, thus cannot figure out that there are already similar label for other groups. The computation speed however increases relatively slow with around 5 seconds for 200 snippets compared to 2 seconds for 100 snippets.



All groups (155)
- Document Clustering (9)
- 1997 015 (9)
- Linux Clustering (10)
- Clustering Services (6)
- Software For Clustering (2)
- High Availability Clustering (12)
- Data Clustering (8)
- Browsers Will Redirect (2)
- Text Clustering (9)
- Clustering (6)
- Text Clustering (5)
- Workload Characterization (1)
- Text Clustering (8)
- Clustering Stateless Session (3)
- Management (1)
- Linux Clustering (6)
- Conceptual Clustering (10)
- High Availability Clustering (6)
- Data Clustering (9)
- Linux (2)
- Clustering Mindmaping (2)
- Machines (3)
- Clustering Algorithms (4)
- Other (22)

All groups (247)
- Results (14)
- Text Clustering (8)
- High Availability Clustering (14)
- Document Clustering (4)
- Windows (7)
- White Papers (19)
- Matlab (3)
- Browsers Will Redirect (2)
- Clustering (2)
- Windows (4)
- Management (1)
- Data Clustering Engine (6)
- Xserve For Clustering (2)
- Linux (2)
- Linux (6)
- Clustering White Papers (9)
- Clustering White Papers (6)
- Linux Clustering (17)
- Web (15)
- Data Clustering (9)
- Innovations And Patents (7)
- Clustering Microsoft Clustering (6)
- Districts (1)
- Mysql Clustering Upcoming (2)
- Clustering Software (6)
- Other (75)

Figure 6.14: Example of clustering results produced by TRC for query "clustering" with different number of snippets : 100 (left) and 200 (right).



All groups (133)
- Jaguar Drivers Club (10)
- Jaguar Drivers Club (4)
- Club (16)
- Announces The Release (2)
- Bumper (3)
- Jaguar Cars (9)
- Auto (11)
- Apple Previews Jaguar (8)
- Wildlife Conservation Society (2)
- Jaguar Technologies (3)
- Panthera Onca 35k (5)
- Parts (7)
- Friendly Page Send (2)
- Jaguar (5)
- Other (46)

All groups (138)
- Jaguar Drivers Club (10)
- Auto (11)
- Jaguar Panthera Onca (6)
- Jaguar Racing (2)
- Jaguar Drivers Club (4)
- Parts (9)
- Announces The Release (2)
- Server With Jaguar (4)
- British Rock Band (4)
- Jaguar Cars (16)
- Jaguar Austria (2)
- Website Van Jaguar (2)
- Jaguar Technologies (3)
- Panthera Onca 35k (5)
- Classic Jaguar (5)
- Jaguar Owners (3)
- Apple Previews Jaguar (6)
- Atari Jaguar Faq (5)
- Jaguar Cars (9)
- Jaguar Films (3)
- Other (27)

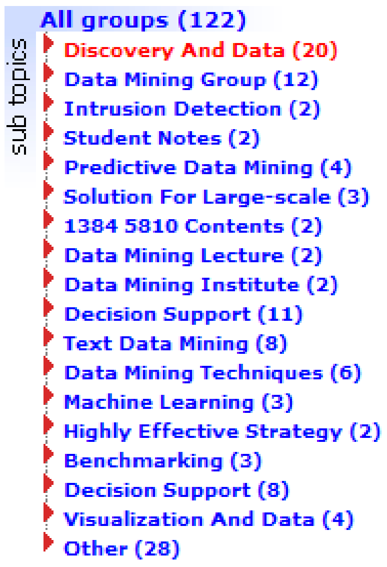
All groups (144)
- Jaguar Drivers Club (10)
- Apple Previews Jaguar (6)
- Auto (12)
- Jaguar Parts (8)
- Atari Jaguar Faq (4)
- Bumper (3)
- Jaguar Racing (2)
- Jaguar Films (3)
- Club (16)
- Announces The Release (2)
- Server With Jaguar (4)
- Quark (3)
- Jaguar Owners (3)
- Wildlife Conservation Society (2)
- Jaguar Austria (2)
- Jaguar Association Germany (2)
- Website Van Jaguar (2)
- Jaguar Technologies (3)
- Endangered Description (1)
- Sportsbook (2)
- Panthera Onca 35k (5)
- Classic Jaguar (5)
- Friendly Page Send (2)
- Jaguar Cars (9)
- Jaguar Panthera Onca (6)
- Atari Jaguar Faq (5)
- Other (22)

Figure 6.15: Example of clustering results produced by TRC for query "jaguar" with number of initial clusters set at 20 (left), 30 (center) and 40 (right).

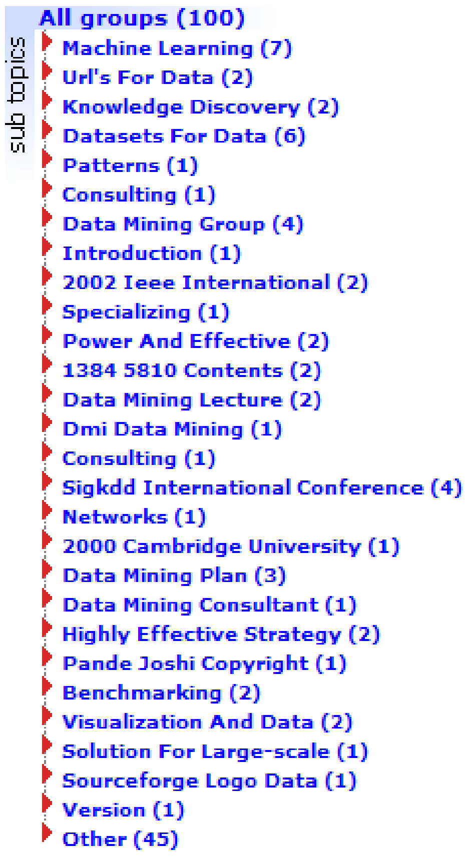
6.6.3. Number of initial clusters

As all K-means based algorithms, TRC is very sensitive to the number of initial clusters. Setting this number right for specific text collection is the point for success. Experiments show that the higher value for initial clusters, the better are the results.

6.6.4. Similarity threshold



All groups (122)
- Discovery And Data (20)
- Data Mining Group (12)
- Intrusion Detection (2)
- Student Notes (2)
- Predictive Data Mining (4)
- Solution For Large-scale (3)
- 1384 5810 Contents (2)
- Data Mining Lecture (2)
- Data Mining Institute (2)
- Decision Support (11)
- Text Data Mining (8)
- Data Mining Techniques (6)
- Machine Learning (3)
- Highly Effective Strategy (2)
- Benchmarking (3)
- Decision Support (8)
- Visualization And Data (4)
- Other (28)



All groups (100)
- Machine Learning (7)
- Url's For Data (2)
- Knowledge Discovery (2)
- Datasets For Data (6)
- Patterns (1)
- Consulting (1)
- Data Mining Group (4)
- Introduction (1)
- 2002 Ieee International (2)
- Specializing (1)
- Power And Effective (2)
- 1384 5810 Contents (2)
- Data Mining Lecture (2)
- Dmi Data Mining (1)
- Consulting (1)
- Sigkdd International Conference (4)
- Networks (1)
- 2000 Cambridge University (1)
- Data Mining Plan (3)
- Data Mining Consultant (1)
- Highly Effective Strategy (2)
- Pande Joshi Copyright (1)
- Benchmarking (2)
- Visualization And Data (2)
- Solution For Large-scale (1)
- Sourceforge Logo Data (1)
- Version (1)
- Other (45)

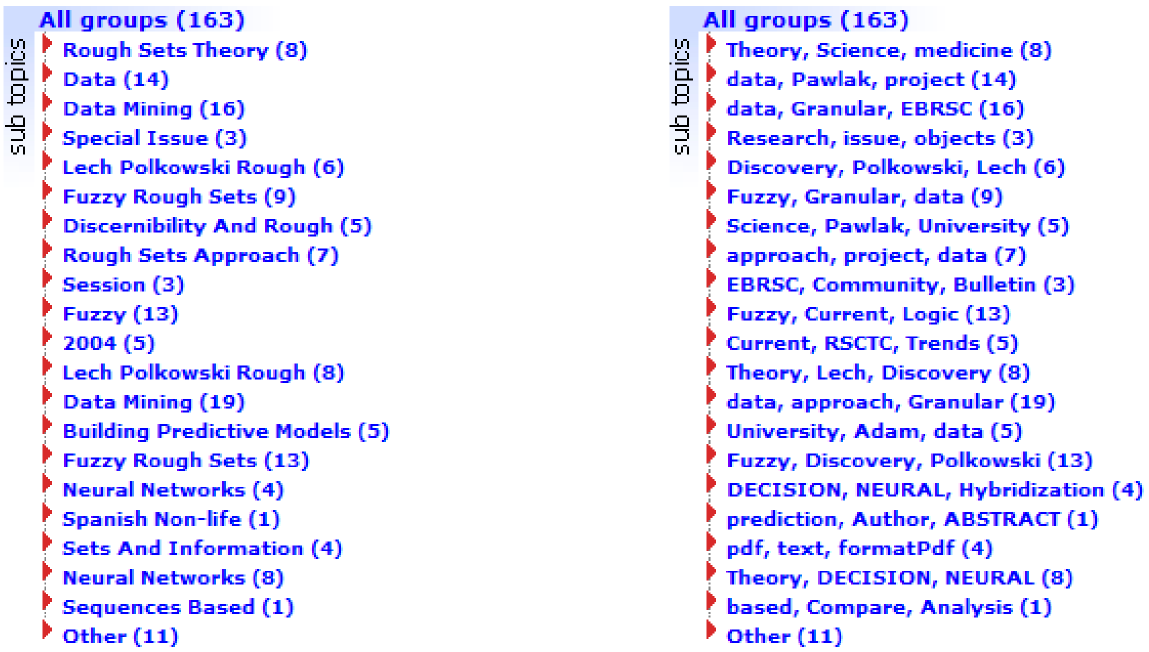
Figure 6.16: Example of clustering results produced by TRC for query "data mining" with number of similarity threshold of 0.25 (left) and 0.5 (right).

Similarity threshold determines if snippet is assigned to a cluster. Higher value will produce more precise group, while setting it too low can results in incoherent groups. We have experimentally estimated that a threshold between range 0.2 and 0.5 are most reasonable.

In this example (see Figure 6.16), setting similarity to 0.5 seems to high resulting in lots of tiny over-specialized groups (one-elemental) and also leaves a large portion of documents unclassified (in "Others" group). With 0.25, cluster produces are more readable and exhibits most important topics in this query.

6.6.5. Phrase usage

It is generally believed [44, 19, 40] that phrases can carry more meanings and therefore are better candidates than single words. The example below shows the clusters with label generated from phrases in comparison with labels formed from "heaviest" term/words in cluster.


sub topics

All groups (163)
- Rough Sets Theory (8)
- Data (14)
- Data Mining (16)
- Special Issue (3)
- Lech Polkowski Rough (6)
- Fuzzy Rough Sets (9)
- Discernibility And Rough (5)
- Rough Sets Approach (7)
- Session (3)
- Fuzzy (13)
- 2004 (5)
- Lech Polkowski Rough (8)
- Data Mining (19)
- Building Predictive Models (5)
- Fuzzy Rough Sets (13)
- Neural Networks (4)
- Spanish Non-life (1)
- Sets And Information (4)
- Neural Networks (8)
- Sequences Based (1)
- Other (11)

sub topics

All groups (163)
- Theory, Science, medicine (8)
- data, Pawlak, project (14)
- data, Granular, EBRSC (16)
- Research, issue, objects (3)
- Discovery, Polkowski, Lech (6)
- Fuzzy, Granular, data (9)
- Science, Pawlak, University (5)
- approach, project, data (7)
- EBRSC, Community, Bulletin (3)
- Fuzzy, Current, Logic (13)
- Current, RSCTC, Trends (5)
- Theory, Lech, Discovery (8)
- data, approach, Granular (19)
- University, Adam, data (5)
- Fuzzy, Discovery, Polkowski (13)
- DECISION, NEURAL, Hybridization (4)
- prediction, Author, ABSTRACT (1)
- pdf, text, formatPdf (4)
- Theory, DECISION, NEURAL (8)
- based, Compare, Analysis (1)
- Other (11)

Figure 6.17: Clusters of query "rough sets" with label generated from phrases (left) and single words (right).

It is clearly visible that phrases are much better at conveying overall topics structure. It also seems to be better in describing cluster's contents. Our approach while based on simple n-gram generation techniques was able to produce good intuitive quality phrases.

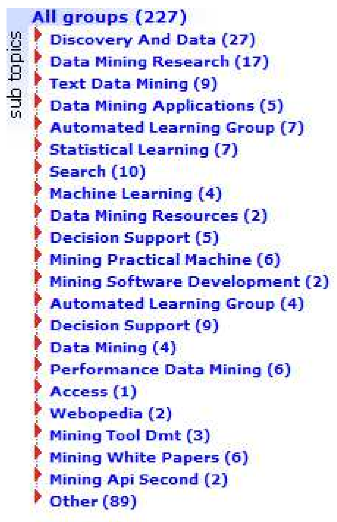
6.6.6. Comparison with other approaches

This part provides a comparison of TRC with other algorithms like LINGO [19], AHC [40], STC [39] and Vivisimo [36]. All algorithms (included our own TRC) were developed within the Carrot2 Framework [37]. This has made possible to compare algorithms running in the same environment and the results could be actually put side-by-side (as it is in Figure 6.18). 200 snippets results of query "data mining" was used as input for 5 algorithms, each of them executed using their default parameters.

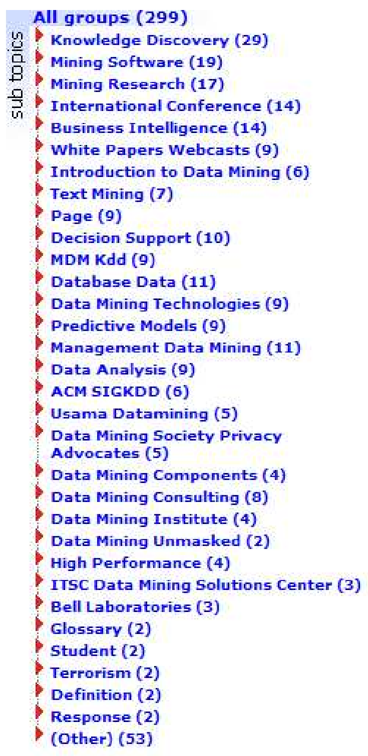
We will compare algorithms, focus on distinguish features of each.

LINGO has very interesting describe-first algorithm providing one of the best labelling amongst all algorithms. It lacks of hierarchical clusters seems to be the only draw back of this approach. The processing speed is comparable to TRC. Due to the natural matching of document to label, for the first sight, the quality of cluster seems good although is it hard to quantify it without details evaluation using standard measures (and not by "eyes"). Group of "Others" documents seems to be large although smaller than TRC.

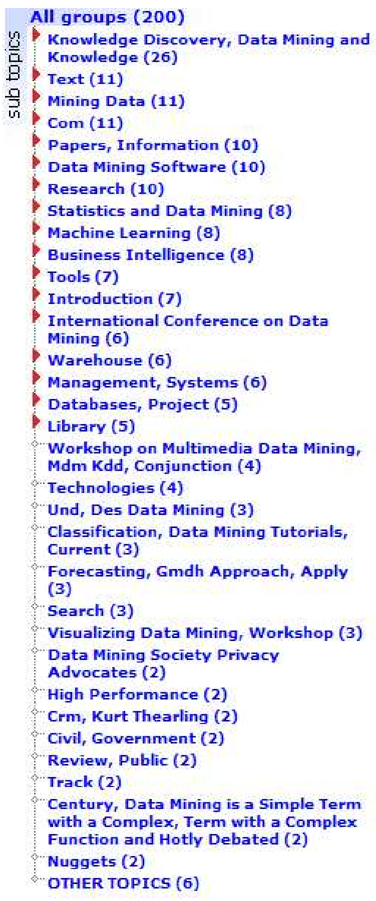
AHC This algorithm produces hierarchies and uses label merging/pruning to produce reasonable good labels. It tends to have several single words label which is less descriptive.



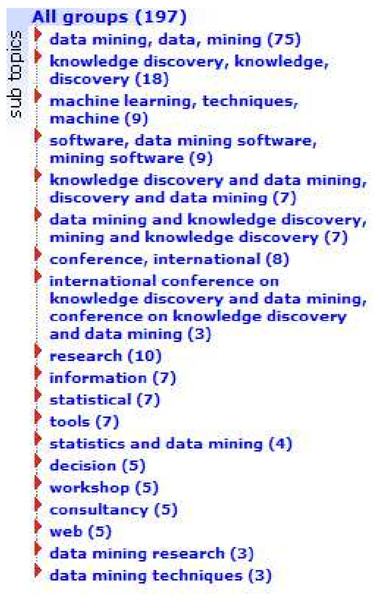
All groups (227)
- Discovery And Data (27)
- Data Mining Research (17)
- Text Data Mining (9)
- Data Mining Applications (5)
- Automated Learning Group (7)
- Statistical Learning (7)
- Search (10)
- Machine Learning (4)
- Data Mining Resources (2)
- Decision Support (5)
- Mining Practical Machine (6)
- Mining Software Development (2)
- Automated Learning Group (4)
- Decision Support (9)
- Data Mining (4)
- Performance Data Mining (6)
- Access (1)
- Webopedia (2)
- Mining Tool Dmt (3)
- Mining White Papers (6)
- Mining Api Second (2)
- Other (89)



All groups (299)
- Knowledge Discovery (29)
- Mining Software (19)
- Mining Research (17)
- International Conference (14)
- Business Intelligence (14)
- White Papers Webcasts (9)
- Introduction to Data Mining (6)
- Text Mining (7)
- Page (9)
- Decision Support (10)
- MDM Kdd (9)
- Database Data (11)
- Data Mining Technologies (9)
- Predictive Models (9)
- Management Data Mining (11)
- Data Analysis (9)
- ACM SIGKDD (6)
- Usama Datamining (5)
- Data Mining Society Privacy Advocates (5)
- Data Mining Components (4)
- Data Mining Consulting (8)
- Data Mining Institute (4)
- Data Mining Unmasked (2)
- High Performance (4)
- ITSC Data Mining Solutions Center (3)
- Bell Laboratories (3)
- Glossary (2)
- Student (2)
- Terrorism (2)
- Definition (2)
- Response (2)
- (Other) (53)



All groups (200)
- Knowledge Discovery, Data Mining and Knowledge (26)
- Text (11)
- Mining Data (11)
- Com (11)
- Papers, Information (10)
- Data Mining Software (10)
- Research (10)
- Statistics and Data Mining (8)
- Machine Learning (8)
- Business Intelligence (8)
- Tools (7)
- Introduction (7)
- International Conference on Data Mining (6)
- Warehouse (6)
- Management, Systems (6)
- Databases, Project (5)
- Library (5)
- Workshop on Multimedia Data Mining, Mdm Kdd, Conjunction (4)
- Technologies (4)
- Und, Des Data Mining (3)
- Classification, Data Mining Tutorials, Current (3)
- Forecasting, Gmdh Approach, Apply (3)
- Search (3)
- Visualizing Data Mining, Workshop (3)
- Data Mining Society Privacy Advocates (2)
- High Performance (2)
- Crm, Kurt Thearling (2)
- Civil, Government (2)
- Review, Public (2)
- Track (2)
- Century, Data Mining is a Simple Term with a Complex, Term with a Complex Function and Hotly Debated (2)
- Nuggets (2)
- OTHER TOPICS (6)



All groups (197)
- data mining, data, mining (75)
- knowledge discovery, knowledge, discovery (18)
- machine learning, techniques, machine (9)
- software, data mining software, mining software (9)
- knowledge discovery and data mining, discovery and data mining (7)
- data mining and knowledge discovery, mining and knowledge discovery (7)
- conference, international (8)
- international conference on knowledge discovery and data mining, conference on knowledge discovery and data mining (3)
- research (10)
- information (7)
- statistical (7)
- tools (7)
- statistics and data mining (4)
- decision (5)
- workshop (5)
- consultancy (5)
- web (5)
- data mining research (3)
- data mining techniques (3)

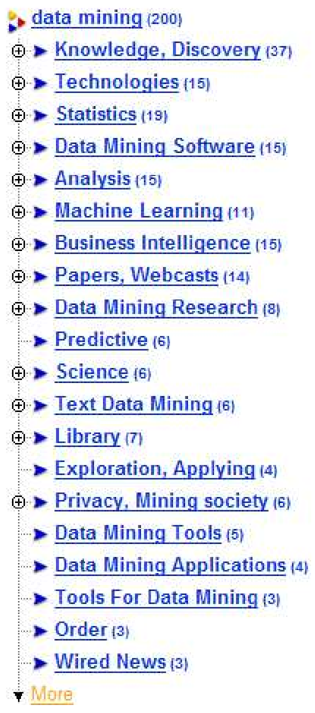

data mining (200)
- Knowledge, Discovery (37)
- Technologies (15)
- Statistics (19)
- Data Mining Software (15)
- Analysis (15)
- Machine Learning (11)
- Business Intelligence (15)
- Papers, Webcasts (14)
- Data Mining Research (8)
- Predictive (6)
- Science (6)
- Text Data Mining (6)
- Library (7)
- Exploration, Applying (4)
- Privacy, Mining society (6)
- Data Mining Tools (5)
- Data Mining Applications (4)
- Tools For Data Mining (3)
- Order (3)
- Wired News (3)

Figure 6.18: Comparison of results for a query "data mining" produced by different algorithm. From left to right: TRC, LINGO, AHC and STC and Vivisimo. All output excepts Vivisimo were produced using *Carrot*^{2} visualization component.

As a hierarchical agglomerative technique, processing time of this algorithm is the longest amongst all approaches.

STC It was implemented along with the creation of the *Carrot*^{2} framework. It's experimental character and also being a pioneer, definitely put him in other light when comparing with other approaches. While its cluster contents are quite good, it lacks of good descriptive label although using phrases. It seems to suffer from the same problem that TRC sometime has, duplicated labels.

Vivisimo Being a commercial application, its algorithm is not published. It seems to have the best cluster labels, supporting also hierarchies.

TRC Our approach by visual comparison doesn't have any distinguish features. Its cluster label quality can be comparable to AHC, approaching LINGO. It suffers from one problem, namely large "Others" group of unclassified documents. Using near-neighbor post-processing helps reduce this slightly. However, our algorithm produce, as a side-effects, groups of related terms – tolerance classes – that could be used/stored additionally as a dynamic thesaurus. Its abilities to enhance document representation using upper approximation also benefits in cluster contents discovery, although this needs to be quantified in more detailed experiments in the future.

6.6.7. Strength and weaknesses

This section will discuss strengths and weaknesses and its outcomes on clustering quality. There are several aspects that we feel worth noticed in our approach :

Use of Tolerance Rough Set for representation enhancement The generated tolerance class demonstrates real semantic relatedness while based on simple co-occurrences in snippets. Upper approximations of documents have been proved to enrich inter-snippets and snippets-cluster relation.

Soft/overlapping clusters In text domain, it is natural that documents can have diverse vocabulary and are related to more than one topic. TRC used K-means with soft assignment places document in many clusters, each with calculated degree of membership.

Use phrases for cluster label Phrase has proven to better characterize cluster's contents than single words. We utilized n-gram based approach to generate phrases, from which most significant within cluster are chosen as label.

Extensibility Although TRC is designed and implemented as one component within *Carrot*^{2} Framework for performance reason, it is composed from set of pieces, interacting with each other through clearly defined interfaces. Therefore, different implementations can be plugged in easily, without modifying the whole architecture.

TRC was designed and implemented as a proof-of-concept, to apply the ideas of Tolerance Rough Set to search results clustering. As results, not all of its aspects have been well-developed. Some of its deficiencies are:

1. TRC uses a version of K-means, which consequently limits it to produce only flat partitions.

2. Also because of usage of K-means without sophisticated initialization, the number of clusters must be given a priori. This can be a serious limitation as text collection even with the same size may present different vocabulary diversities, which algorithm at present state cannot adapt to.

3. The K-means approach adapted by TRC seems to struggle with boundaries region, parts of clustering space that is sparse and cannot be classified to any groups. This results in a so called the "Others" documents, a relatively large group of documents that doesn't match to any clusters.

Chapter 7

Conclusion and future work

During the last ten year, the rapid development of information technology and the rise of the internet and the Web have revolutionized the way people use and access information. We have from the age of information deficiency into the state of information overload. This has given birth to a new, exiting domain of research referred to as Web Mining or Web Information Retrieval. The most well known product/service resulting from these researches are web search engines, starting from the once famous AltaVista [4], to today's *de facto* standard, Google. Search engines have been found useful in so many everyday activities and have become so ubiquitous that many people today can hardly imagine their work and life without it. While search engines have achieved quite good results in delivering answer for well formulated, precise queries, they have been less efficient as a tool for discovering and exploration of broad ranged topics. Post-processing search results using clustering algorithm may be the solution for the problem.

This thesis was thought as a proof-of-concept demonstration of Tolerance Rough Set application to web mining domain. We wanted to investigate how rough set theory and its ideas, like approximations of concept, could be practically applied in a task of search results clustering. The result is a design of a TRC — a Tolerance Rough Set Clustering algorithm for search results. We also decide to implement the proposed solution within an open-source framework, *Carrot*^{2}. During our work, we have found that for a successful application in user-oriented task such as search results clustering required emphasis not only on the quality of cluster contents but also it description. Therefore, we have adapted an n-gram based approach to generate cluster label from more descriptive phrases.

The experiment we have carried has showed that Tolerance Rough Set and upper approximation it offers can indeed improve the representations, which have positive effects on clustering quality. The results are promising and encourage further work on its application. Due to the limited amount of time allocated for this thesis we could only manage to provide a simple expert evaluation and comparison with other algorithms. However, we feel that the application has fulfilled its promises and has proven the usefulness of usage of rough sets theory.

Worth emphasis is also the fact that implementation of algorithm presented in this thesis, including all source codes, will be contributed to the *Carrot*^{2} project and will be available at http://carrot2.sourceforge.net. We hopes that this will foster further experiments and enhancements to the algorithm to be made, not only by the author but also other researchers.

7.1. Further possible improvement

We are aware that the proposed algorithm, while has shown promises and produce encouraging results, still has many limitations. Therefore, we present possible ideas that are worth considering for algorithm improvement.

- For K-means algorithm, identifying initial clusters is a very important aspect that can have great influences on overall results. Thus, employing more sophisticated technique for identifying initial cluster based on overall data set can be beneficial.
- Vocabulary of text documents exhibit concepts that can be naturally categorized into hierarchies. Ability to generate hierarchy of concepts is therefore demanded. TRC at the present state could be improved to create hierarchical clustering by divide-and-conquer — dynamically divide two big clusters or iteratively merging smaller ones.
- The natural side-effect of TRC is the generation of tolerance classes of terms that can be thought of as concept. At present these classes are generated per request and discarded after clustering. We could memorize generated classes into some kind of context-sensitive (per query) thesaurus. Subsequent queries that are related to past request can use this thesaurus for vocabulary enhancement or label generation. Machine learning techniques could then be used to learn from that thesaurus memory.

Bibliography

[1] Adam Schenker, Mark Last, A. K. Design and implementation of a web mining system for organizing search engine results. In *Data Integration over the Web (DIWeb), First International Workshop, Interlaken, Switzerland, 4 June 2001.* (2001), pp. 62–75.

[2] Adrzej Skowron, J. S. Tolerance approximation spaces. *Fundamenta Informaticae 27*, 2-3 (1996), 245–253.

[3] AllTheWeb. http://alltheweb.com.

[4] Altavista. http://altavista.com.

[5] Baeza-Yates, R., and Ribeiro-Neto, B. *Modern Information Retrieval*, 1st ed. Addison Wesley Longman Publishing Co. Inc., May 1999.

[6] Chakrabarti, S. *Mining the Web.* Morgan Kaufmann, 2003.

[7] Cutting, D. R., Karger, D. R., Pedersen, J. O., and Tukey, J. W. Scatter/gather: a cluster-based approach to browsing large document collections. In *Proceedings of the 15th annual international ACM SIGIR conference on Research and development in information retrieval* (1992), pp. 318–329.

[8] David J. Hand, Heikki Mannila, P. S. *Principles of Data Mining (Adaptive Computation and Machine Learning).* MIT Press, 2001.

[9] Excite. http://excite.com.

[10] Google. http://google.com.

[11] Hearst, M. A., and Pedersen, J. O. Reexamining the cluster hypothesis: Scatter/gather on retrieval results. In *Proceedings of SIGIR-96, 19th ACM International Conference on Research and Development in Information Retrieval* (Zürich, CH, 1996), pp. 76–84.

[12] Infoseek. http://infoseek.com.

[13] Jain, A. K., Murty, M. N., and Flynn, P. J. Data clustering: a review. *ACM Comput. Surv. 31*, 3 (1999), 264–323.

[14] Jan Komorowski, Zdzislaw Pawlak, L. P. A. S. Rough sets: a tutorial, 1998.

[15] Jansen, B. J., Spink, A., Bateman, J., and Saracevic, T. Real life information retrieval: a study of user queries on the web. *ACM SIGIR Forum 32*, 1 (1998), 5–17.

[16] Jiawei Han, M. K. *Data Mining: Concepts and Techniques*, 1st ed. Morgan Kaufmann, 2000.

[17] KARTOO. http://kartoo.com.

[18] LARSEN, B., AND AONE, C. Fast and effective text mining using linear-time document clustering. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining* (1999), pp. 16–22.

[19] OSINSKI, S. An algorithm for clustering of web search result. Master's thesis, Poznan University of Technology, Poland, June 2003.

[20] PAWLAK, Z. *Rough sets: Theoretical aspects of reasoning about data.* Kluwer Dordrecht, 1991.

[21] PAWLAK, Z., GRZYMALA-BUSSE, J., SLOWINSKI, R., AND ZIARKO, W. Rough sets. *Communications of the ACM 38*, 11 (1995), 88–95.

[22] PIROLLI, P., SCHANK, P., HEARST, M., AND DIEHL, C. Scatter/gather browsing communicates the topic structure of a very large text collection. In *Proceedings of the SIGCHI conference on Human factors in computing systems* (1996), pp. 213–220.

[23] PORTER, M. F. An algorithm for suffix stripping. In *Readings in Information Retrieval*, P. W. Karen Sparck Jones, Ed. Morgan Kaufmann, San Francisco, 1997, pp. 130–137.

[24] PROJECT, T. O. D. http://dmoz.org/.

[25] SALTON, G. *Automatic text processing: the transformation, analysis, and retrieval of information by computer.* Addison-Wesley Longman Publishing Co., Inc., 1989.

[26] SAORI KAWASAKI, NGOC BINH NGUYEN, T. B. H. Hierarchical document clustering based on tolerance rough set model. In *Principles of Data Mining and Knowledge Discovery, 4th European Conference, PKDD 2000, Lyon, France, September 13-16, 2000, Proceedings* (2000), D. A. Zighed, H. J. Komorowski, and J. M. Zytkow, Eds., vol. 1910 of *Lecture Notes in Computer Science*, Springer.

[27] SILVERSTEIN, C., HENZINGER, M., MARAIS, H., AND MORICZ, M. Analysis of a very large altavista query log. Tech. Rep. 1998-014, Digital SRC, 1998.

[28] SMADJA, F. A. From n-grams to collocations: An evaluation of xtract. In *29th Annual Meeting of the Association for Computational Linguistics, 18-21 June 1991, University of California, Berkeley, California, USA, Proceedings* (1991), pp. 279–284.

[29] SPINK, A., WOLFRAM, D., JANSEN, B., AND SARACEVIC, T. Searching the web: The public and their queries, 1998.

[30] STEFANOWSKI, J., AND WEISS, D. Carrot^2 and language properties in web search results clustering. In *Proceedings of AWIC-2003, First International Atlantic Web Intelligence Conference* (Madrid, Spain, 2003), E. M. Ruiz, J. Segovia, and P. S. Szczepaniak, Eds., vol. 2663 of *Lecture Notes in Computer Science*, Springer, pp. 240–249.

[31] STEINBACH, M., KARYPIS, G., AND KUMAR, V. A comparison of document clustering techniques. In *KDD Workshop on TextMining* (2000).

[32] SYSTEMS, A. http://maps.map.net/.

[33] T. Honkela, S. Kaski, K. L., and Kohonen, T. Websom–self-organizing maps of document collections. In *Proceedings of WSOM'97, Workshop on Self-Organizing Maps, Espoo, Finland, June 4-6 1997, Helsinki University of Technology, Neural Networks Research Centre, Espoo, Finland.* (1997), pp. 310–315.

[34] Tu Bao Ho, N. B. N. Nonhierarchical document clustering based on a tolerance rough set model. *International Journal of Intelligent Systems 17*, 2 (2002), 199–212.

[35] van Rijsbergen, C. J. K. *Information Retrieval.* Butterworths, London, 1979.

[36] Vivisimo. http://vivisimo.com/faq/technology.html.

[37] Weiss, D. *Carrot2 Developers Guide.* http://www.cs.put.poznan.pl/dweiss/carrot/site/developers/ma

[38] Weiss, D. A clustering interface for web search results in polish and english. Master's thesis, Poznan University of Technology, Poland, June 2001.

[39] Weiss, D. A clustering interface for web search results in polish and english, 2001.

[40] Wroblewski, M. A hierarchical www pages clustering algorithm based on the vector space model. Master's thesis, Poznan University of Technology, Poland, July 2003.

[41] Xu, J., and Croft, W. B. Query expansion using local and global document analysis. In *Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval* (1996), pp. 4–11.

[42] Xu, J., and Croft, W. B. Improving the effectiveness of information retrieval with local context analysis. *ACM Trans. Inf. Syst. 18*, 1 (2000), 79–112.

[43] Zamir, O., and Etzioni, O. Web document clustering: A feasibility demonstration. In *Research and Development in Information Retrieval* (1998), pp. 46–54.

[44] Zamir, O., and Etzioni, O. Grouper: a dynamic clustering interface to web search results. *Computer Networks (Amsterdam, Netherlands: 1999) 31*, 11–16 (1999), 1361–1374.

[45] Zdzislaw Pawlak, A. S. Rough membership functions. In *Advances in the Dempster-Shafer Theory of Evidence* (New York, 1994), J. K. M. Fedrizzi and R. Yager, Eds., John Wiley & Sons, pp. 251–271.

[46] Zhang, D. *Towards Web Information Clustering.* PhD thesis, Southeast University, Nanjing, China, January 2002.

[47] Zhao, Y., and Karypis, G. Criterion functions for document clustering: Experiments and analysis, 2001.

[48] Ziarko, W. Variable precision rough set model. *Journal of Computer and System Sciences 46-1*, 11 (1993), 39–55.