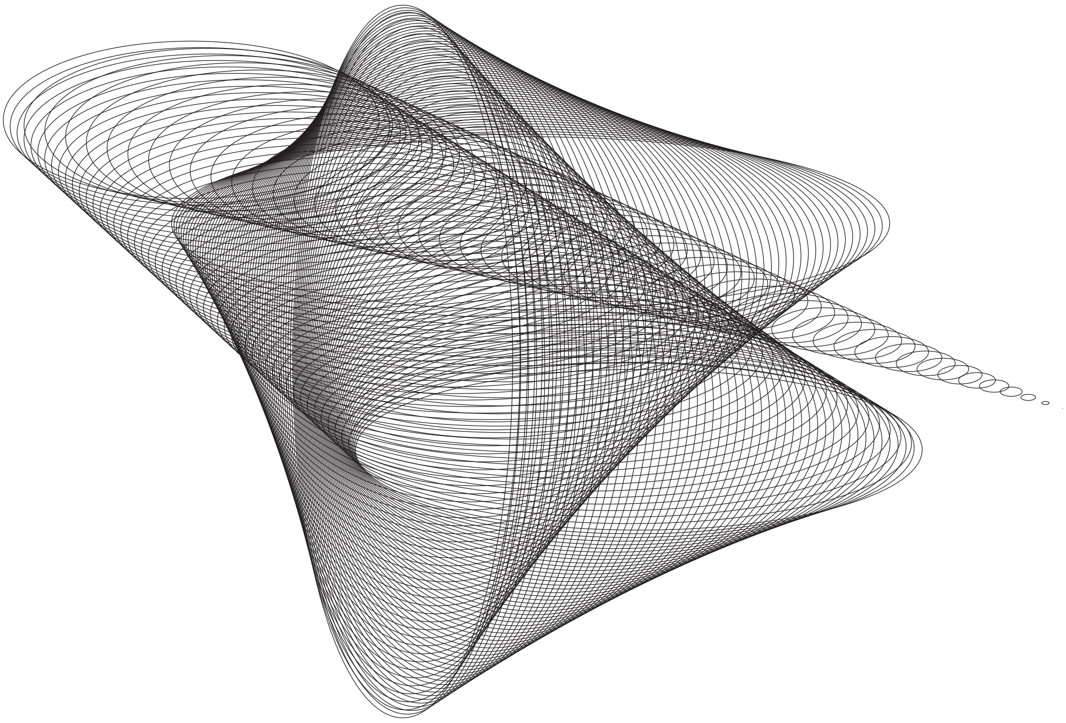
MASTER'S THESIS

KENNETH LOLK VESTER
MOSES CLAUS MARTINY

Information Retrieval in Document Spaces Using Clustering



DTU Department of Informatics and
Mathematical Modelling
Technical University of Denmark

August 2005

In collaboration with:
mondosoft

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 4525 3351, Fax +45 4588 2673
reception@imm.dtu.dk
`www.imm.dtu.dk`

In collaboration with:

mondosoft

Mondosoft A/S
Vestergade 18 E
DK-1456 Copenhagen K, Denmark
Phone +45 7020 2009, Fax +45 7020 2509
info@mondosoft.com
`www.mondosoft.com`

IMM-THESIS: ISSN 1601-233X

"Information is the oxygen of the modern age.
It seeps through the walls topped by barbed wire,
it wafts across the electrified borders."

– Ronald Reagan

Abstract

Today, information retrieval plays a large part of our everyday lives – especially with the advent of the World Wide Web. During the last 10 years, the amount of information available in electronic form on the Web has grown exponentially. However, this development has introduced problems of its own; finding useful information is increasingly becoming a hit-or-miss experience that often ends in information overload. In this thesis, we propose document clustering as a possible solution for improving information retrieval on the Web.

The primary objective of this project was to assist the software company Mondosoft in evaluating the feasibility of using document clustering to improve their information retrieval products. To achieve this end, we have designed and implemented a clustering toolkit that allows experiments with various clustering algorithms in connection with real websites.

The construction of the toolkit was based on a comprehensive analysis of current research within the area. The toolkit encompasses the entire clustering process, including data extraction, various preprocessing steps, the actual clustering and postprocessing. The aim of the document clustering is finding similar pages and, to a lesser degree, search result clustering of webpages. The toolkit is fully integrated with Mondosoft's search engine and utilises a two-stage approach to document clustering, where keywords are first extracted and then clustering is performed using these keywords.

The toolkit includes prototype implementations of several promising algorithms, including several novel ideas/approaches of our own. The toolkit implements the following 5 clustering algorithms: K-Means, CURE, PDDP, GALOIS and a novel extended version of Apriori. In addition to this, we introduce two novel approaches for extracting n-grams and a novel keyword extraction scheme based on Latent Semantic Analysis.

To test the capabilities of the implemented algorithms, we have subjected them to extensive performance tests, both in terms of memory and computational requirements. Our tests clearly show that CURE and GALOIS become infeasible in connection with larger websites (10,000+ pages). To evaluate the quality of the remaining three algorithms and the toolkit in general, we have also performed a user test based on the similar pages found by the algorithms.

The user test shows with statistic significance that the quality of the algorithms for this task can be ranked in the following order: Apriori, K-Means and PDDP. Furthermore, the test provided evidence that both K-Means and Apriori were as good as or better than a search-based approach for finding similar pages. Finally, we have found strong evidence that the LSA-based method for keyword extraction is better for subsequent clustering, than pure truncation of terms based on their local weight.

Preface

This thesis is submitted in partial fulfillment of the requirements for the Master of Science degree in Engineering (M.Sc. Eng.) at the Technical University of Denmark (DTU), Kongens Lyngby, Copenhagen. The authors are Moses Claus Martiny and Kenneth Lolk Vester. The project supervisor was Jan Larsen from the Department of Informatics and Mathematical Modelling at DTU. The thesis was made in collaboration with Mondosoft A/S.

Project work was done at Mondosoft in Copenhagen during the Spring and Summer of 2005.

Acknowledgements

Before we commence presenting our work, we would like to extend our thanks to the people who made it possible.

First, everyone at Mondosoft - none mentioned, none forgotten - for their invaluable help and inspiration during the project.

Our supervisor Jan Larsen for keeping us on track and providing us with help, inspiration and encouragement throughout the project.

Dr. Irwin King from the Chinese University of Hong Kong for providing the initial spark of inspiration for this project and for his help during the project.

Our test participants for taking the time to provide us with the test data necessary for assessing the quality of our algorithms.

Finally, we would like to thank our teachers at DTU and elsewhere for providing us with the tools necessary to undertake a project of this magnitude.

Kongens Lyngby, August 11, 2005

Kenneth Lolk Vester Moses Claus Martiny

Contents

Part I

Main Report

Chapter 1

Introduction

This thesis represents the written part of our Master's Thesis project, which was undertaken in collaboration with the software company Mondosoft[1] that produces a suite of information retrieval products. The project concerns different approaches to document clustering of medium to large websites.

The primary objective of the project is to help improving Mondosoft's range of information retrieval products. We have focused on assisting Mondosoft in evaluating the quality and feasibility of using document clustering to this end. In order to achieve this, we have designed a clustering toolkit that implements prototypes of promising clustering approaches. We have used this toolkit for performing tests and experiments with clustering.

Our main contributions in this project include the following methods/algorithms:

- A trie-based approach to discovering groups of stemmed words.
- Keyword extraction using Latent Semantic Analysis.
- A method for extracting n-grams using the results from a lattice-based clustering.
- A method for extracting n-grams using behaviour tracking data from a search engine.
- An extended version of the data mining algorithm, Apriori, which is able to build Galois lattices.

Below, we will briefly outline the structure of this thesis, to give the reader a better overview of the content herein.

1.1 Structure of the Report

First, in **Chapter 2 - Background and Scope**, we outline the background for the project, giving a brief introduction to information retrieval in general and document clustering in particular. We then continue with a brief introduction to Mondosoft and their products. Having presented

[1] `http://www.mondosoft.com/`

the reader with this background information, we present the scope and problem definition of the project.

In **Chapter 3 - Analysis and Literature Study**, we discuss the results of the comprehensive literature study that forms the basis for the project. First, we give a brief overview of the information retrieval process that clustering is part of. We then move on to discussing different approaches to document clustering. After this, we introduce and discuss *Latent Semantic Analysis*, which is a promising statistical and mathematical method for analysis of textual information. Then, we discuss various preprocessing techniques that can be used to improve the quality and efficiency of document clustering. Finally, we briefly define and discuss the applications of document clustering that we have chosen to focus on.

In **Chapter 4 - Chosen Approach** we discuss the two-stage approach to document clustering that we have chosen as basis for the clustering toolkit. We then motivate our choice of clustering algorithms to implement. Finally, we outline the architecture that we have designed for the toolkit.

In **Chapter 5 - Implemented Preprocessing** we discuss the preprocessing steps we have implemented in the clustering toolkit. First, we briefly describe how we have integrated the clustering toolkit with MondoSearch™. Then we discuss the implemented term filtering, quickly moving on to discussing the implemented term stemming and weighting schemes. Finally, we discuss the two novel schemes for bigram extraction that we have designed and implemented.

In **Chapter 6 - Implemented Keyword Extraction Algorithms** we discuss the implemented algorithms for keyword extraction. Here we mainly focus on the novel approach based on LSA that we have conceived and implemented. Finally, we briefly outline a simpler approach based on truncation.

In **Chapter 7 - Implemented Clustering Algorithms** we introduce and discuss the 5 chosen clustering approaches. For each algorithm, we also touch upon the algorithm's complexity, advantages and weaknesses. Finally, we provide important and relevant implementation details for each algorithm.

In **Chapter 8 - Implemented Postprocessing** we discuss our implementation of the chosen postprocessing schemes that use the implemented clustering to improve the information retrieval process.

In **Chapter 9 - Performance of the Implemented Algorithms** we try to assess the running-time and memory consumption of the implemented algorithms (in connection with large websites) using an actual website as test data.

In **Chapter 10 - Toolkit Evaluation** we discuss some of the experiences we have gained, while implementing and experimenting with the clustering toolkit. First, we present some of the challenges that we have encountered when working with websites instead of pure text data. Then we move on to a sensitivity analysis, outlining how sensitive the system is to changes in different parameters. Finally, we present some preliminary results of using search result clustering in connection with ambiguous queries.

In **Chapter 11 - User Test** we outline the user test that we have carried out to determine how our algorithms compare to each other and to a simpler search-based approach with regard to finding similar pages. We then move on to presenting the findings of the test.

Finally, in **Chapter 12 - Conclusion and Future Work** we close this thesis with a conclusion, where we summarise our contributions and main findings. In addition to the conclusion, we have included our recommendations to Mondosoft for the work ahead as well as a section that outlines future research opportunities within the areas of this thesis including future perspectives for document clustering in general.

Chapter 2

Background and Scope

In this chapter, we will give a brief introduction to information retrieval in general, including common ways of representing documents in an information retrieval system. We will then move on to document clustering, where we will briefly define and introduce the area. This includes common applications, a taxonomy of document clustering and an overview of the challenges specific to this area of information retrieval.

We will also give a brief introduction to Mondosoft, their products and motivations for participating in this project. Finally, after having introduced the basic terminology of clustering and Mondosoft's situation, we will define the scope of the project based on the aims of Mondosoft and ourselves.

2.1 A Brief Introduction to Information Retrieval

Information Retrieval (IR) is an emerging subfield of information science concerning representation, storage, access and retrieval of information [FBY92, BYRN99]. Current research areas within the field of IR include:

- Searching and querying
- Ranking of search results
- Navigating and browsing information
- Optimising information representation and storage
- Document classification (into predefined groups)
- Document clustering (into automatically discovered groups)

Information retrieval dates more than 4000 years back to the beginning of written language [BYRN99], as information retrieval is related to knowledge stored in textual form. Today text has grown to become:

> "... the primary way that human knowledge is stored, and after speech, the primary way it is transmitted."[1]

Traditionally, information retrieval was a manual process, mostly happening in the form of book lists in libraries, and in the books themselves, as tables of contents, other indices etc. These lists/tables usually contained a small number of index terms (e.g. title, author and perhaps a few subject headings) due to the tedious work of manually building and maintaining these indices.

The above was true through most of history up until the middle of the 20th century, where the digital computer fundamentally changed the way that Man was able to store, search and retrieve textual information [FBY92]. As a result, information retrieval has grown well beyond its previous limited form, mostly concerned with indexing and searching books and other kinds of textual information [BYRN99].

Today, information retrieval plays a much larger part of our everyday lives – especially with the advent of the Internet, and the World Wide Web (the Web) in particular. During the last 10 years, the amount of information available in electronic form on the Web has grown exponentially. Almost any kind of desired information is available on the Web, including: Bibliographic collections, news and message files, software libraries, multimedia repositories, online encyclopedias, commercial information etc. [CR95]. Or, as pictured in [BYRN99, p. 2]:

> "the web is becoming a universal repository of human knowledge and culture, which has allowed an unprecedent [sic] sharing of ideas and information in a scale never seen before"

Furthermore, the amount of documents managed in organisational intranets that represent the accumulated knowledge of the organisations is also quickly growing, and efficient access to (and retrieval of) these documents has become vital to the success of modern organisations [BEX02].

Information retrieval is at the center stage of this "revolution" and is a necessary condition for its continuing expansion into even more areas of our lives. However, the Web and related technologies have introduced problems of their own – finding useful information is increasingly becoming a hit-or-miss experience that often ends in information overload. People still find it difficult (if not impossible) to consistently locate and retrieve information relevant to their needs. As Roussinov and Chen in [RC01] pessimistically put it:

> "Our productivity in generating information has exceeded our ability to process it, and the dream of creating an information-rich society has become a nightmare of information overload."

[1] [FBY92, p. vii]

Search engines (such as Google and Yahoo) that are the common gateways to the huge collections of electronic text on the Web, are continuously optimised and enhanced to better serve the needs of their users. It has been recognised that the low precision[2] of search results in search engines today is the major limiting factor for locating relevant information [RG00]. An example[3] of this is a user searching for "computer games", wanting to know more about the underlying technologies. However, even with the sophisticated ranking algorithms used in today's search engines, it is not possible to predict (based on this query) whether the user is interested in information in the latest advances in game technology or if the user is simply searching for the latest entertainment products.

In [RG00], Rüger and Gauch outlines 3 basic approaches to amending this problem:

- Cluster documents to allow users to better preview and navigate the information structure of the returned results.
- Organise documents into a predefined hierarchy of categories, where the user benefits from familiar terms when navigating to the right information (e.g. Yahoo).
- Learn more about the user's interests/tasks in order to allow the system to automatically identify documents that are relevant for this particular user or for the task at hand.

This report is mainly concerned with research into the first of the above proposed solutions; document clustering.

In modern information retrieval systems, several models exist to represent the information contained in a large collection of textual documents. Below, we have outlined the two most common models, known as the Boolean model and the vector model.

2.1.1 The Boolean Model

The Boolean model for information retrieval is a simple retrieval model based on set theory and Boolean algebra. In its essence, the boolean representation of a document is a set of terms, where the terms are words from the document extracted using different measures such as filtering (see section 3.3.1). Looking at a collection of documents, each term set would then be represented by a binary/boolean vector where a 1 represents a term present in the document and a 0 a term which is not.

Searching for documents then proceeds by taking a query formulated in Boolean terms and applying it on the index terms of some set of documents. For instance the query for documents with the term k_a and either the term k_b or not the term k_c would result in a selection as demonstrated in figure 2.1.

Using such Boolean expressions for information retrieval has the great advantage of being easy to learn and very simple to implement. The distance measure of the Boolean model both between queries and documents and between documents in general is thus simply the size of the

[2] Precision refers to the common metric defined as the number of relevant documents in the result compared to the total number of documents in the result.

[3] Adapted from [RG00].

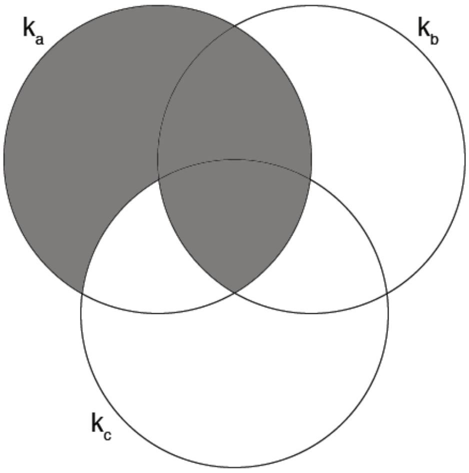


Figure 2.1: *The selection resulting from the query* $q = k_a \wedge (k_b \vee \neg k_c)$.

intersection of the term sets (i.e. the dot-product of the vectors) – the greater the intersection, the closer (more similar) the document is to the query.

However, the Boolean model also has a number of disadvantages. First of all, whether a document is relevant for the query or not has become a binary decision – there is no grading scale for relevance. Boolean queries further mean that partial matching (i.e. if the document fulfills some, but not all requirements) is impossible and thus, ranking becomes difficult if not also impossible, when strictly using Boolean algebra.

And finally there is no clear way to translate information needs to a boolean expression directly, since a user request often is more like this: "I want documents about A, where from these, I am most interested in B, less in C whereas D could also interest me if E is fulfilled". Thus, even though the language is as simple as it is, it still requires the users to be able to structure and express their needs in a different way than they normally would.

A number of alternative schemes countering the above drawbacks have been designed, where one of the often used models is the vector model [BYRN99] which we will describe in the next section.

2.1.2 The Vector Model

The vector model for information retrieval is built on a thorough analysis of the documents: By analysing the words or terms in the documents and comparing with the overall use of these words, each word can be assigned a value describing its relative significance – either to the containing document or to the set of documents (the collection). This value assignment is called *index term weighting* or simply *term weighting*, which will be elaborated in section 3.3.3.

In this way, by adding weights to the index terms, we can suddenly view the problem from a different perspective. Where we (in the Boolean model) used to consider documents as sets of terms, we can now consider them as vectors of terms. This adds a geometrical "dimension" to our model, where we can calculate query relevance and document similarity based on geometrical distance in a multi-dimensional[4] *term* or *feature space* . Such geometrical distance measures could for instance be the Cosine or the Euclidean distance – both are measures that work for an arbitrary number of dimensions.

Using the vector model means that we now have several options for information retrieval. Instead of the "either the document is relevant or not" approach of the Boolean model, we have gained a "partial matching" approach, where documents can match the query even if just one term in the query is present. We can, of course, still process the term space using set operations.

2.1.3 Other Models

The above models are by far the easiest to comprehend and thus to use for information retrieval purposes. The last of the "classic" models is the probabilistic model, which is somewhat more complicated, and not necessarily better than the vector model [BYRN99], so we have left it from the discussion.

All of the classical models suffer from the fact that they assume that all index terms are mutually independent, but a simple example can in fact prove that this assumption is not always correct: If document A contains the term "computer" and document B contains the term "kindergarten", would you then consider discovering the term "network" in document A just as likely as in document B?

This suggests that the classical models, where the term axes are all mutually orthogonal might need some modifications, we introduce *Latent Semantic Analysis* in section 3.2 as a possible solution to this. Another approach to this is to use language-based models, where word order and linguistic information play a more important role.

2.2 Document Clustering

Imagine that you were given 100 newspaper articles and asked to sort them in a number of piles, reflecting their content. The number of piles and the central themes of the article piles are entirely up to you. You are also free to choose whether you want to read through every article or if you will only read the headings and skim through the contents. Such is the task of a document clustering system, with the only difference being that the task involves a lot more than 100 documents and is to be performed automatically by a computer.

We define[5] document clustering as:

> The automatic discovery of document clusters/groups in a document collection, where the formed clusters have a high degree of association (with regard to a given similarity measure) between members, whereas members from different clusters have a low degree of association.

In other words, the goal of a good document clustering scheme is to minimise intra-cluster

[4] The many terms each become an axis in the space, which results in very high dimensionality.

[5] Adapted from [FBY92].

distances between documents, while maximising inter-cluster distances (using an appropriate distance measure between documents). A distance measure (or, dually, similarity measure) thus lies at the heart of document clustering. Several ways for measuring the similarity between two documents exist, some are based on the vector model (e.g. Cosine distance or Euclidean distance) while others are based on the Boolean model (e.g. size of intersection between document term sets). More advanced approaches exist, for instance using Latent Semantic Analysis to transform the vector space into a space of reduced dimensionality.

Clustering is sometimes erroneously referred to as automatic classification, however, this is inaccurate, since the clusters found are not known prior to processing (as the name "classification" would imply) [FBY92].

2.2.1 The Cluster Hypothesis

The application of clustering in information retrieval is largely based on the *cluster hypothesis*, which was originally postulated by van Rijsbergen and Jardine in 1971. In [TVR02] Tombros, Villa and van Rijsbergen state the hypothesis in this way:

> "[The cluster hypothesis] states that relevant documents [to a given information need] tend to be more similar to each other than to non-relevant documents, and therefore tend to appear in the same clusters."

This of course implies that the information retrieval process can be improved by somehow helping the user to discover the cluster(s) that contain most of the relevant documents. This is precisely what most applications of document clustering in information retrieval systems try to accomplish.

2.2.2 Applications of Document Clustering

Generally, clustering is used in statistics to discover the structure of large "multivariate" data sets. It can often reveal latent relationships hidden in complex data.

Within information retrieval, clustering (of documents) has several promising applications, all concerned with improving efficiency and effectiveness of the retrieval process. Some of the more interesting include:

- **Finding Similar Documents** to a given document. This feature is often used when the user has spotted one "good" document in a search result and wants *more-like-this*. The interesting property here is that clustering is able to discover documents that are conceptually alike in contrast to search-based approaches that are only able to discover whether the documents share many of the same words.

- **Search Result Clustering** allowing the user to get a better overview of the documents returned as results in the search, and to navigate towards clusters that are relevant to the user's information need.

- **Guided/Interactive Search**, where clustering is used to help the user drill down and find the desired information step-by-step by gradually refining the search.
- **Organising Site Content into Categories** allowing browsing of the site in a Yahoo-like fashion.
- **Recommender System** that, based on the documents the user has already visited, recommends other documents. A typical use of this is in an e-commerce setting, where products that might interest the customer are suggested based on products the user has already examined/bought.
- **Faster/Better Search** utilising the clustering to optimise the search. A user query could for instance be compared to clusters instead of the individual documents, effectively limiting the search space.

2.2.3 A Taxonomy of Clustering Methods

When treating the subject document clustering, it is always important to know which kinds of clustering are necessary and feasible for a given application. In this section, we will describe the most obvious clustering classifications in order to later better describe the clustering methods we choose to implement and to justify our choices.

Hard vs. Soft Clustering

Sometimes, information can be relevant for several categories at once. A subject such as “biomedical engineering” could be relevant for categories such as “chemical engineering”, “biology” and “medicine” all at once. A clustering method able to cluster documents in several categories at once is called a “soft” clustering method, since the boundaries of the clusters may be thought of as *soft*. This concept is illustrated in figure 2.2, where documents appear in more than one cluster.

More often than not - information is not categorised this way. Take for instance the way libraries organise information - it is unthinkable that a book on biomedical engineering would be placed at three different places in the library (even if it possessed three such books).

Thus, information might instead be organised according to the category it fits the best. The book on biomedical engineering might therefore be placed in the chemical engineering category since the book has the most in common with other books in this category. This is called *hard* clustering and is illustrated in figure 2.3 where it is seen that the documents closest together are clustered.

Hierarchical vs. Flat Clustering

When organising information, it is of course always important to determine what kind of organisation is necessary. For some applications, simply putting all data into “buckets” and returning these buckets on request is an acceptable solution. This is practical and realistic, when the inter-cluster distances are high, the intra-cluster distances low, the noise of the term space negligible

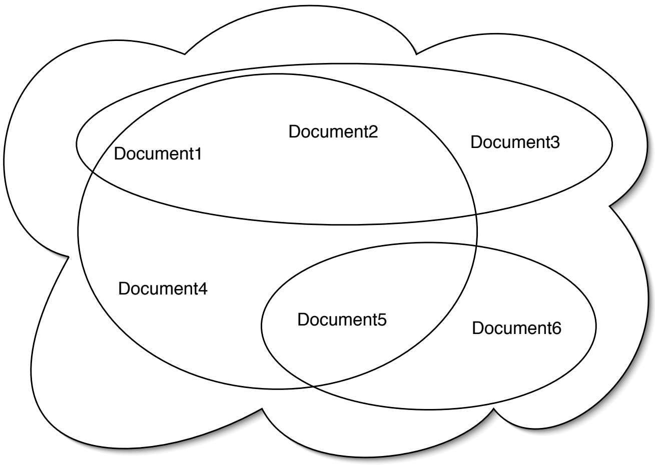


Figure 2.2: *Documents clustered using soft clustering.*

and all clusters of approximately the same (limited) size. Clustering methods simply resulting in all the documents of the document space ending in different buckets are called *flat* clustering methods.

However, information can often be regarded from many different angles, clusters might be of varying size and each cluster may separately be considered as a subspace with its own clusters. If we try to represent this with a flat clustering, unrelated information may be considered as being in the same context and put in the same cluster. To avoid this, information can also be organised *hierarchically.* The top-most levels of such a hierarchy would contain the largest clusters, which should also be the most general topics of the collection. The large clusters are then split into several smaller clusters as we proceed down the hierarchy. A hierarchical clustering method thus produces a term space clustering and a hierarchical structure for the clustering.

Hierarchical clustering may be seen as a special case of "soft" clustering, where the clusters contain other clusters. However, to make the distinction between soft and hard relevant in connection with hierarchical clustering, we have chosen to define *soft hierarchical clustering* as a hierarchical clustering, where there is overlap between clusters on the same "level" of the hierarchy. A *hard hierarchical clustering* is then the case, where there is no overlap between clusters on the same level. As we shall see in the next chapter, several approaches to creating a hierarchical document clustering exist.

Online vs. Offline Clustering

Another important consideration is when clustering is to be performed. This depends on many factors such as the size of the term space to cluster, the complexity of the algorithm to be used, what the clustering is meant to be used for and how many of such operations must be performed at the same time.

If the clustering operations require processing a huge term space, *offline* clustering is probably the most suitable approach. This means creating and storing the clusters in a fast database and

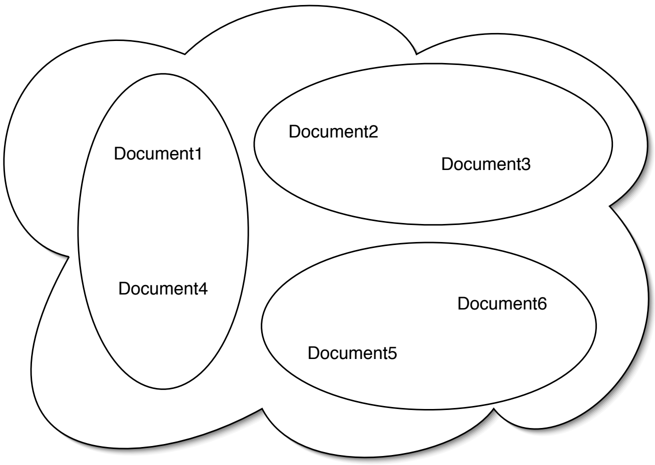


Figure 2.3: *Documents clustered using hard clustering.*

only leaving simple operations to be performed when querying the database (postprocessing). This of course, has the drawback of not being up-to-date as soon as a single document is changed or added, and thus requires frequent updates.

Online clustering, on the other hand, requires no such updates hence the fact that all operations are performed on request. This obviously requires very fast algorithms and a limited data set, but might be practical for clustering search results (see section 3.4).

2.2.4 Challenges in Document Clustering

Although commercial information retrieval systems[6] utilising clustering exist, document clustering is far from a trivial or solved problem. The clustering process is filled with challenges like[7]:

- Selecting appropriate features of the documents that should be used for clustering.
- Selecting an appropriate similarity measure between documents.
- Selecting an appropriate clustering method utilising the above similarity measure.
- Implementing the clustering algorithm in an efficient way that makes it feasible in terms of required memory and CPU resources.
- Finding ways of assessing the quality of the performed clustering.
- Finding feasible ways of updating the clustering if new documents are added to the collection.
- Finding ways for applying the clustering to improve the information retrieval task at hand.

[6] See for instance `http://www.vivisimo.com/`

[7] Adapted from [FBY92, chap. 16]

Furthermore, with medium to large document collections (10,000+ documents), the number of term-document relations is fairly high (millions+), and the computational complexity of the algorithm applied is thus a central factor in whether it is feasible for real-life applications. If a dense matrix is constructed to represent term-document relations, this matrix could easily become too large to keep in memory - e.g. $100,000$ documents × $100,000$ terms = 10^{10} entries $\sim$ 40 GB using 32-bit floating point values. If the vector model is applied, the dimensionality of the resulting vector space will likewise be quite high (10,000+). This means that simple operations, like finding the Euclidean distance between two documents in the vector space, become time consuming tasks.

In order to somewhat amend this, the sparse nature of term-document relations should be utilised to save the information in what is known as a *sparse matrix*. In this matrix, only non-zero entries and their position within the matrix are stored.

The Curse of Dimensionality

The high-dimensional space mentioned above also has another drawback referred to as the *curse of dimensionality*: Since the high-dimensional space is very sparsely populated, two randomly picked points in a hyper-cube tend to have a constant distance from each other, regardless of the distance measure applied [RG00]. Few meaningful clusters thus exist in such a space. In addition, the sparsity of the document space leads to many documents being orthogonal (sharing none of the same features).

It is thus important to use methods of reducing dimensionality in connection with document clustering. Promising methods range from simple filtering to more advanced approaches, like *Latent Semantic Analysis*[8].

2.3 Scope of the Project

This project was initiated as a collaboration between ourselves (the authors) and the software company Mondosoft[9]. The main aim of the project is to help improving the quality and effectiveness of Mondosoft's information retrieval products through clustering utilising cutting-edge research within the area. We envision that our partnership with Mondosoft will be mutually beneficial, since we offer Mondosoft a fresh perspective and an academic focus, while Mondosoft, on the other hand, allows us to utilise their substantial knowledge and experience within information retrieval as a foundation for our work.

First, we will briefly introduce Mondosoft's basis for this project, including their main products and their motivation and expectations for the project. After this, we will present the problem definition that we have developed in collaboration with Mondosoft.

[8] See section 3.2

[9] See `http://www.mondosoft.com/`

2.3.1 Introduction to Mondosoft

Mondosoft is a software company offering a suite of enterprise search, analytics and site optimisation products. Mondosoft has three main product offerings:

- MondoSearch™ is a multi-lingual search engine targeted at search within individual (large) sites.
- BehaviorTracking™ is an analytical reporting tool providing information on the search activity and visitor behaviour on the site.
- InformationManager™ utilises the data from BehaviorTracking™ to allow web managers to refine content and search experience.

As illustrated in figure 2.4, the three main products thus form a kind of "feedback loop" allowing BehaviorTracking™ data to influence how MondoSearch™ works (via InformationManager™).

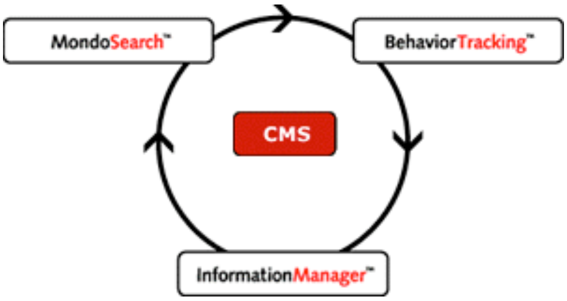


Figure 2.4: *Mondosoft's three main products form a feedback loop.*

Compared with global search engines such as Google and Yahoo, Mondosoft is in an interesting position. The site owners that manage Mondosoft's search products are the owners of the site contents, on which the search engine is operating. This means that Mondosoft is as concerned with the needs of the information providers (site owners) as with the needs of the information users (users visiting the site). This is a challenge, but fortunately, the needs of these two groups often coincide (e.g. "if the customer can't find it, the customer can't buy it).

Mondosoft's interest in clustering is primarily to provide:

- A "More-Like-This" feature that allows users to expand a search by finding documents similar to an interesting document in the search result.
- An automatic (hierarchical) clustering/categorisation of the search results to help users getting a better overview of the returned pages, with the clear aim of improving the search experience and effectiveness. At the moment, MondoSearch™ uses manually assigned categories to partition search results, but experience shows that most site owners would prefer a more automatic process.

However, since Mondosoft is always interested in improving search quality, effectiveness and experience, they are also very interested in other applications for clustering that might help achieve this.

Since the typical customers of MondoSearch™ crawl and index their sites at regular intervals (e.g. once a week) – Mondosoft believes that it would be natural if the clustering happened as part of this regular process (i.e. offline clustering).

2.3.2 Problem Definition

In short, the scope of this project is to help Mondosoft to evaluate the quality and feasibility of using cutting edge clustering methods to hopefully improve their products. In order to achieve this, we will design a clustering toolkit that implements prototypes of the most promising clustering methods. This toolkit should be aimed at clustering experiments on medium to large-scale websites already indexed by Mondosoft's MondoSearch™.

For practical reasons, we will focus on clustering websites in English. However, whereever possible, we will try to create language-independent solutions. The clustering algorithms that will be implemented in the toolkit should be based on current academic research and theory. If feasible, the implemented clustering should produce soft clusters as we believe that soft clusters better capture the underlying information structure in a website.

With the above toolkit, we will carry out experiments primarily aimed at evaluating the quality and feasibility of using clustering in connection with a "More-Like-This" feature, finding other documents similar to a chosen document. Time allowing, we will also try to evaluate automatic categorisation of search results. If possible, the evaluation should be based on user tests, since we strongly believe that, at the end of the day, the user experience is what really matters. In addition to our own experiments, we envision that Mondosoft will be able to utilise our toolkit for further experiments with other clustering applications, and as a foundation for integrating clustering-based functionality into their products.

Hence, the primary goal of the toolkit is to create practically viable (in terms of space and time requirements) implementations of the clustering algorithms, which can be used for offline clustering of the document collection in connection with crawling and indexing a website using MondoSearch™. The clustering toolkit must therefore be closely integrated with MondoSearch™ – for instance using data structures similar to those of MondoSearch™ making later integration into a final product easier. If possible, the toolkit should also utilise the information recorded by Mondosoft's BehaviorTracking™ to improve clustering.

Chapter 3

Analysis and Literature Study

It is important to emphasise that getting from a collection of documents to a clustering of the collection, is not merely a single operation, but is more a process in multiple stages. These stages include more traditional information retrieval operations such as crawling, indexing, weighting, filtering etc. Some of these other processes are central to the quality and performance of most clustering algorithms, and it is thus necessary to consider these stages together with a given clustering algorithm to harness its true potential. To help the reader better understand the context that the actual clustering is a part of, we will therefore give a brief overview of the clustering process, before we begin our literature study and analysis.

We have divided the offline clustering process into the four stages outlined below, each stage possibly having multiple substages:

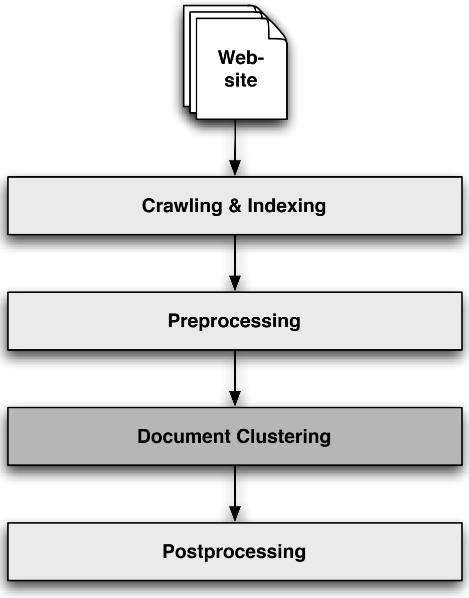


Figure 3.1: *Document Clustering is just one stage in a multi-stage process.*

Crawling and Indexing Crawling is the process where the links of a given set of websites are traversed to gather all relevant pages. As part of this process, the information of the individual pages is indexed. Often, some information is filtered out (crawl-time filtering) for instance pictures and known stop-words. Indexing is responsible for building indexes of term/document relations. Such indexes form the basis for the term-document matrix that is often the starting point for clustering.

Preprocessing Various subprocesses concerned with adapting the above indexes for clustering purposes are what we refer to as "preprocessing techniques". These include everything from the basic task of converting the indexes into a suitable data representation (e.g. a term-document matrix) to more advanced techniques such as various kinds of filtering, stemming and term weighting.

Document Clustering This is the actual document clustering process concerned with discovering clusters of documents using some given distance measure. This process is of course the main focus of the report.

Postprocessing The actual applications of a document clustering to some purpose within information retrieval is what we refer to as "postprocessing". In this project, our postprocessing efforts are mainly focused on a more-like-this function and, to a lesser degree, clustering of search results.

Fortunately, crawling and indexing is a part of the foundation MondoSearch™ provides us with. Hence, we do not need to spend time "reinventing the wheel". On the other hand, the remaining three stages are where we will focus our efforts in this project.

Below, we will first discuss different approaches to document clustering. Then, we will look into *Latent Semantic Analysis*, an emerging technique within the field of information retrieval that has been applied within both traditional search and clustering with quite promising results. Finally, we will examine various preprocessing techniques, and also briefly discuss more-like-this and search result clustering.

3.1 Common Clustering Methods

Over the course of our literature study, we have encountered a wide variety of different approaches to document clustering. We have classified the most common approaches into the below four "main" categories, inspired by [ZHTY03]. The classification is primarily based on the underlying techniques that the algortihms rely on, but also on the kinds of clusters the algorithms produce (see our taxonomy in section 2.2.3).

For each class of algorithms, we have attempted to assess its feasibility for the scope of this project. Feasibility concerns include:

- Computational requirements – we should be able to cluster medium to large websites, within a reasonable amount of time.

- Memory requirements – our clustering toolkit should be able to run on a standard PC limited by a 32-bit address space.
- Required input data – the data the algorithm requires should be easily extractible from MondoSearch™ (or the BehaviorTracking™ database). Algorithms should thus primarily rely on term-document relations and not the raw documents.
- Evidence that the algorithms actually work – algorithms without promising published results demonstrating the feasibility are thus less interesting.
- Complexity of the approach – for two algorithms showing similar qualities, we prefer the simpler algorithm (a kind of "Occam's razor" for clustering algorithms).

3.1.1 Partition-based Clustering

Partition-based algorithms partition a document collection into a number of hard clusters using a given distance measure. These methods are traditionally divided into *single-pass* methods and *relocation-based* methods. Common for these two classes of methods are that they usually work well for finding spherically shaped clusters in small to medium-sized databases. However, these methods are less adept at discovering clusters of non-convex shapes.

A good partitioning follows the general cluster criterion that documents in the same cluster are close and related, while documents in different clusters are different and apart (based on the given distance measure).

Single-pass Partitioning

Single-pass algorithms use a greedy approach assigning each document to a cluster only once. They are thus very fast, but at the cost of quality, since the single-pass approach does not allow for correction of "mistakes" (documents assigned to a suboptimal cluster).

The most simple implementation works by attempting to find spherical clusters of equal size by assigning the first document to a new cluster. Subsequent documents are then either (A) assigned to an existing cluster, or (B) used to create a new cluster if the distance to the closest cluster[1] exceeds a predefined threshold. [FBY92, chap. 16] [ZHTY03]

Relocation-based Partitioning

Relocation-based algorithms run in multiple passes. First, an initial partitioning is formed, and thereafter an iterative relocation technique is applied to attempt to improve the partitioning by moving documents/objects between partitions.

The most popular relocation-based algorithm is by far *K-Means* (see section 7.1). K-Means has almost become a "gold" standard that many other clustering algorithms are measured by. In K-Means K random documents are chosen as initial cluster centers. Hereafter, all documents

[1] "Closest" is defined depending on the given algorithm - this can for instance be a centroid or a representative document for the cluster.

in the collection are assigned to the closest cluster[2]. Then, the cluster centers are recalculated and the documents are relocated. The process of calculating centers and reassigning documents is repeated until the clustering has been stabilised.

Several variations of K-Means exist, for instance *K-centroid* (see [ZHTY03]), where the document closest to the mean of the cluster is chosen to represent the cluster as a kind of centroid. Common for all K-Means variations is that they usually perform quite well with regard to quality and have reasonable computational requirements.

3.1.2 Hierarchical Clustering

Hierarchical clustering approaches attempt to create a hierarchical decomposition of the given document collection thus achieving a hierarchical structure as described in section 2.2.3. Hierarchical methods are usually classified into *Agglomerative* and *Divisive* methods depending on how the hierarchy is constructed.

Agglomerative Methods

Agglomerative methods start with an initial clustering of the term space, where all documents are considered representing a separate cluster. The closest clusters using a given inter-cluster similarity measure are then merged continuously until only 1 cluster or a predefined number of clusters remain. Some of these methods are more suitable for discovering clusters of non-convex forms than partition-based algorithms. Agglomerative methods normally produce hard (hierarchical) clusters.

Agglomerative algorithms are usually classified according to the inter-cluster similarity measure they use. The most popular of these are *single-link*, *complete-link* and *group average*. More exotic similarity measures, for instance *Ward's method* (see [FBY92]) also exist. Common for all agglomerative methods is high computational complexity, often quadratic or worse.

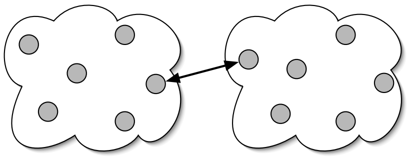


Figure 3.2: *The single-link distance measure.*

Single-Link Clustering algorithms based on this similarity measure join the two clusters containing the two closest documents (using a given distance measure) that are not yet in the same cluster (see figure 3.2). This scheme can be implemented in a reasonably effective way relative to the two below similarity measures. However, it has a slight inclination towards chaining[3]

[2] Here, the closest cluster is the cluster with the closest center using the given distance measure.

[3] Chaining occurs when clusters that should be separate are joined too early in the hierarchy causing strange or elongated clusters in the term space.

[FBY92, chap. 16]. Popular clustering methods based on this similarity measure include *CURE* and *Minimum Spanning Tree* (MST). CURE is particularly interesting since it aims to avoid chaining effects by employing a so-called shrinking scheme.

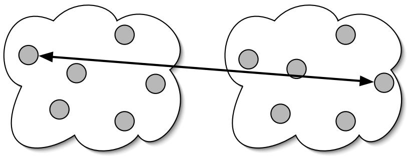


Figure 3.3: *The complete-link distance measure.*

Complete-Link Clustering algorithms using this similarity measure join the two clusters with the minimum "most-distant" pair of documents (see figure 3.3). In this way, clusters are kept small and compact since all documents within a cluster have a maximum distance to each other [FBY92, chap. 16]. The *Vorhees* algorithm is a typical example of a clustering approach using this similarity measure, however, the computational complexity of clustering algorithms using this measure is normally higher than that of single-link based algorithms.

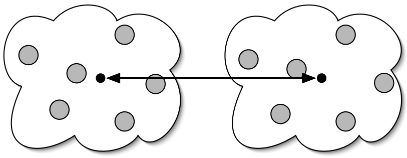


Figure 3.4: *The group average distance measure.*

Group Average Clustering algorithms using this similarity measure join the two clusters with the minimum average document distance – i.e. the distance between cluster centers (see figure 3.4). This similarity measure results in clusters somewhere between the large single-link clusters and the more compact complete-link clusters [FBY92, chap. 16]. The *BIRCH* algorithm is a typical example of a clustering algorithm using this similarity measure, however, experiments in [GRS98] show that CURE in practise outperforms this algorithm.

Divisive Methods

Divisive clustering algorithms start with a single cluster containing all documents. It then continuously divides clusters until all documents are contained in their own cluster or a predefined number of clusters are found. They thus work directly opposite of the agglomerative methods discussed above. These methods are usually significantly faster than the agglomerative methods, but have the drawback that "splits" or divisions cannot be undone to undo erroneous decisions – see appendix C for an example of this. Due to the divisive nature, these algorithms almost always produce a hard (hierarchical) clustering.

Principal Direction Divisive Partitioning (PDDP) [Bol98] is a typical example of a divisive clustering algorithm splitting the documents of the term space based on finding the documents' projection on the principal direction. In addition to this, partition-based algorithms such as K-Means can be made divisive by applying them to the largest cluster of the set of clusters until a preset number of clusters are found.

3.1.3 Keyword-based Clustering

This is not really a class of clustering methods, but more a specific way of choosing the document features that the clustering is applied on. However, some clustering algorithms are only feasible with a limited set of features per document and thus require that a limited number of core-terms/keywords are assigned per document.

Keyword-based clustering works by assigning a set of core-terms (a form of keywords) to each document and uses these terms as features[4] for the subsequent clustering. This is based on the idea that the main topic/theme of a document can be captured in a relatively small set of core-terms or keywords. This idea is related to the Topic-binder hypothesis[5]:

> "A document set which shares a term (or terms) appearing uniquely in the set treats the same topic and thus the documents in the set resemble each other. We call such a term a 'topic binding term' or a 'topic binder'. This is the 'topic binder hypothesis'..."

The use of index terms (keywords) to categorise library books and academic papers is a common practise and has existed long before the emergence of electronic information collections.

Experiments by Schütze and Silverstein in [SS97] show that clustering using a simple truncation of the terms based on their weight in the given document offer significant speed advantages over "full term clustering" while offering comparable cluster quality. Having a small set of keywords for each document thus allows for interesting clustering approaches such as *lattice-based clustering* and *frequent itemset clustering*. Both would be too computationally demanding to run on the full term set for every document.

Furthermore, the assigned keywords form very sparse document vectors that can be used to drastically speed up distance calculations in more traditional clustering algorithms such as the ones discussed above. Filtering out non-keyword terms can also (depending on the keyword extraction method) have the effect of reducing noise, since only terms important to the main topic of the document are allowed to remain.

Frequent Itemset-based Methods

Frequent itemset clustering is based on finding frequent sets of keywords that often occur together in the document collection – an approach often used in data mining. These sets are then used as soft (flat[6]) clusters, since documents sharing one or more keywords are supposed to be related

[4] As previously indicated, features are a generic term used to describe terms or other information extracted from a document with the intent of using it for clustering.

[5] Stated in [TKUE99]

[6] However, as will be explained in section 7.5, the partial ordering of the frequent itemsets actually forms a lattice and is therefore closely related to the lattice methods.

if the keyword extraction scheme is well designed. The distance measure applied in this kind of clustering is hence the inverse of the size of the intersection between the two documents' keyword sets. The clustering method is based on a kind of Boolean model, since only the binary relation between documents and keywords is used in the distance measure. An advantage of this approach is that clusters are easily labelled by the keywords shared by the contained documents.

However, this method depends on a sufficient overlap between the keywords assigned to documents to form relevant clusters. In addition, the method only works with a rather limited number of keywords assigned per document, since the complexity will otherwise be prohibitive for large document collections.

Several algorithms exist to extract frequent itemsets, but the most popular is currently *Apriori* [AS94], which performs quite well, even on large databases (document collections).

Lattice-based Methods

Lattice-based clustering is closely related to the frequent itemset approach above, but constructs a lattice structure of concepts forming a kind of hierarchy with multiple inheritance. This approach is based on *Formal Concept Analysis* and is often described as a form of conceptual clustering. Since a document can be related to several concepts on the same level (in the lattice), the lattice represents a hierarchical soft clustering of the document collection.

The lattices produced by methods from this class are known as Galois lattices or simply concept lattices. The concepts themselves are based on the mathematically defined notion of *Formal Concept*[7] and are in some sense similar to the above frequent itemsets. A formal concept is defined either by an *intent* by an *extent*, or both. The intent is the set of keywords that defines the concept and the extent is exactly the documents that have these keywords in common.

Several methods exist for constructing a Galois lattice from a set of documents with assigned keywords. A popular and relatively fast method for constructing such a lattice is GALOIS [CR96].

3.1.4 Model-based Clustering

Model-based clustering, as we define it[8], is based on hypothesising models for clusters in the document collection, and then for each document finding the cluster whose model the document best fits. These clustering methods can be both soft or hard as well as flat or hierarchical. Model-based methods cover a wide variety of different approaches, but most fall in two major categories: *Statistical methods* and *Neural Network methods* [ZHTY03].

Statistical Methods

Statistical methods hypothesise models for clusters using statistical analysis of the document collection. Besides from this, these methods range from simple methods like *RPCL* to more ad-

[7] See [HSS03].

[8] Based on [ZHTY03].

vanced approaches such as clustering based on *principal component analysis* or the closely related *latent semantic analysis* (LSA) – for instance *probabilistic latent semantic indexing* (PLSI).

Rival Penalized Competitive Learning [KL99] (RPCL), in all its simplicity, works by first selecting a number of random centers in the term space. Hereafter, documents are taken from the collection as samples. The center which is closest to the sampled document is moved towards the document by some fraction of the distance. The second closest center (the rival) “loses” and is “penalised” by moving it a smaller fraction away from the sampled document. When a given number of samples have been processed, all documents are assigned to clusters containing their closest center. According to King [KL99], this method is considerably faster than K-Means, however, the quality of the clusters is somewhat poorer.

Latent semantic analysis, which is closely related to principal component analysis, is a very promising technology for reducing dimensionality of the term space while at the same time removing noise and revealing hidden (latent) information in the document collection. We have found that LSA looks so promising that we have dedicated section 3.2 in the analysis to discuss this technique. An alternative related method for dimensional reduction (projection into a lower dimension subspace) is *independent component analysis*.

Neural Network Methods

Neural networks are designed to emulate the internal functionings of the human brain, more specifically, the neurons. Neural network-based methods rely on either supervised or unsupervised learning to assigning multi-dimensional outputs to multi-dimensional inputs. Neural networks have been applied with great success in image processing and other related areas.

Methods based on supervised learning require a training phase, where inputs with desired outputs are presented to the network. This means that a very consistent model for the clustering is needed already before the clustering is performed, this is thus rather classification than actual clustering.

Unsupervised learning, on the other hand, dynamically learns/designs a model as it processes input data. A typical clustering method based on unsupervised learning neural networks is *Kohonen's self-organizing maps*[9]. These methods produce fairly good results, but are hampered by long processing time [BYSZ02] and are thus unfeasible for medium and larger document collections (1,000+ documents) [BdMAS02].

Neural networks are also sometimes used to reduce dimensionality of the term-document matrix before running more traditional clustering algorithms [DM02].

3.1.5 Other Clustering Methods

In addition to the four above classes of clustering approaches, several other interesting approaches to clustering webpages exist.

Fuzzy clustering is a special kind of soft clustering that assigns a degree of “belonging” for each possible document-cluster relation (the cartesian product of all clusters and all documents).

[9] See for instance [BdMAS02] or [RC01].

There are thus no clearly defined clusters, only fuzzy clusters with "blurred" borders. An example of a fuzzy clustering algorithm is *fuzzy C-Means* (See [Miy03]). While this is an interesting approach, a more conventional clustering will suffice for the kinds of postprocessing that we are interested in.

When dealing with hypertext, it would be obvious to use the linking structure as a basis for document clustering. Several methods for this kind of link-based clustering exist (see for instance [HZDS01]). However, we feel that these kinds of methods fall outside the scope of this project, since we have chosen to focus on clustering based on the textual contents of the documents.

Clustering methods based on *suffix trees* have often been used to cluster search results "on-the-fly". Suffix trees are a widely used data structure (somewhat related to *tries*) used for string matching. When used for clustering, suffix trees are used to find subphrases shared by a group of documents and then to use these as a basis for clustering. The obvious advantage of this approach is that the clusters can be labelled using natural language subphrases [CSB03]. However, the construction of suffix trees relies on access to the full text of the documents which is not readily available from MondoSearch™.

Finally, a class of clustering algorithms, based on a linguistic analysis of the document collection, exists (see for instance [AFMiA00] & [Hul03]). Such an analysis also relies on access to the full text of the documents as well as to tables of word classes and grammatical rules, which makes the approach impractical for this project.

3.2 Using Latent Semantic Analysis (LSA) to Improve Clustering

3.2.1 Introduction to LSA

Latent Semantic Analysis (LSA) is one of the more promising emerging technologies within the area of Information Retrieval. It is a statistical/mathematical method for indexing, retrieval and analysis of textual information and it has been applied within different fields of machine cognition during the last 10-15 years [NPM01].

LSA is quite similar to Principal Component Analysis, which is often used in statistics to analyse data sets composed of highly correlated variables [Ler99]. In its essence it is a clever mathematical technique for uncovering common patterns of word/term usage across a collection of documents (term co-occurrence). Which terms often occur in the same documents and which documents share some of the same terms? [YCCP]. LSA is thus exploiting the obvious property of natural language; words with similar semantic meanings tend to occur together.

Experiments show that LSA have many promising areas of application within Information Retrieval[10]:

- Automatic generation of domain specific synonyms.
- Keyword Extraction from documents, using global information from the entire collection[11].

[10] List adapted from [Gro]

[11] Thus allowing a given document to be tagged with keywords that are not necessarily included in the document.

- Finding sets of similar documents in a given collection (i.e. Document Clustering).
- Finding documents that are similar to a given document or to a given query (set of terms).
- Spam filtering of email[12].

The results of using LSA within the above areas are surprisingly good. Some of the more commonly aknowledged benefits of LSA is noise reduction and dampening of synonymy (several words can have the same meaning) and polysemy (a word can have several meanings) [CTN03]. LSA is thus often used as a solution to the problem of ambiguous terms (e.g. (software) architecture vs. (building) architecture) and the problem of synonyms in information retrieval (e.g. the user searches for "doctor", but the desired documents contain the term "physician"). In this way LSA can uncover latent (hidden) global information in the document collection. This latent information has many interesting real-world applications. It can for instance be utilised in a search engine to retrieve relevant documents that do not share any terms with the query. E.g. LSA would be able to infer that a given document containing the terms "Saddam", "Gulf","Bush" and "War" with high frequency is related to the term "Iraq"– even though it does not contain it. This can be inferred because other documents that contain the above terms with high frequency also contain the term "Iraq" with high frequency. An LSA-based search engine would thus be able to return the document to users searching for "Iraq".

One of the more remarkable and controversial results of using LSA's inference capabilities is a study by Landauer and Dumais [LD97] that shows that their system based on LSA performed as good as average foreign students in TOEFL-tests of English proficiency[13].

Mathematically LSA works by projecting the large multidimensional term-document space down into a subspace of much smaller dimension, "squeezing" similar terms and similar documents together. It is closely related to neural network models, but is based on the mathematical technique known as *Singular Value Decomposition* (SVD). We will discuss SVD in-depth below, but so far suffice to say that SVD is a powerful mathematical matrix decomposition technique closely akin to factor analysis, that performs an "optimal" (in the least squares sense) dimensional reduction on the original term-document vector space [LFL98].

This dimensional reduction is done while preserving as much information as possible about the document/term vectors while collapsing them down into the much smaller LSA-subspace. But of course information is lost in this process, but in this case information loss is a good thing, since most of the information that is lost is noise that was present in the original high-dimensional term-document space. Similarities that were latent (hidden) in the original term-document matrix are also revealed; similar documents/terms are squeezed closer together while dissimilar documents/terms are torn further apart.

LSA takes a term-document matrix as input and hence works more or less directly on any collection of text documents, no annotation or linguistic information is needed. It is thus language independent in the sense that it can be used with any language without modifications.

[12] Apple's popular email client *Mail* for instance uses a variant of LSA to dynamically learn how to distinguish between spam and normal mail based on the textual content of the emails.

[13] Actually Landauer and Dumais even ventured as far as proposing LSA as a model of children's learning, but this falls outside of the scope of this report.

3.2.2 Singular Value Decomposition (SVD)

As discussed above *Singular Value Decomposition* (SVD) is the mathematical matrix decomposition technique that lies at the heart of LSA. SVD is what enables the “optimal” projection of document or term vectors into any desired lower dimensional LSA-subspace. But before we explore the mathematics behind SVD, it is advantageous to have an intuitive understanding of what the technique aims to achieve. [YCCP] use the following instructive analogy to explain in layman’s terms what the SVD algorithm tries to do:

> “Imagine you keep tropical fish, and are proud of your prize aquarium – so proud that you want to submit a picture of it to Modern Aquaria magazine, for fame and profit. To get the best possible picture, you will want to choose a good angle from which to take the photo. You want to make sure that as many of the fish as possible are visible in your picture, without being hidden by other fish in the foreground. You also won’t want the fish all bunched together in a clump, but rather shot from an angle that shows them nicely distributed in the water. Since your tank is transparent on all sides, you can take a variety of pictures from above, below, and from all around the aquarium, and select the best one.
>
> In mathematical terms, you are looking for an optimal mapping of points in 3-space (the fish) onto a plane (the film in your camera). ’Optimal’ can mean many things – in this case it means ’aesthetically pleasing’. But now imagine that your goal is to preserve the relative distance between the fish as much as possible, so that fish on opposite sides of the tank don’t get superimposed in the photograph to look like they are right next to each other. Here you would be doing exactly what the SVD algorithm tries to do with a much higher-dimensional space.”

In the case of LSA, the high-dimensional space discussed in the quote above is the term-document space, typically having thousands of dimension. This n-dimensional vector space is defined by the row or column vectors[14] of the term-document matrix (cf. the vector space model discussed in section 2.1.2). SVD decomposes the original term-document matrix into three matrices, that factored together reproduce the original matrix:

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \tag{3.1}$$

SVD is based on the following theorem in linear algebra:

> “Any $M \times N$ matrix $\mathbf{A}$ whose number of rows M is greater than or equal to its number of columns N, can be written as the product of an $M \times N$ column-orthogonal matrix $\mathbf{U}$, an $N \times N$ diagonal matrix $\mathbf{\Sigma}$ with positive (or zero) elements (*singular values*), and the transpose of an $N \times N$ orthogonal matrix $\mathbf{V}$. The matrices $\mathbf{U}$ and $\mathbf{V}$ are each orthogonal in the sense that their columns are orthonormal.”[15]

[14] Often an approach where documents are represented as vectors in a term space of dimension N (N being the number of terms) is used, but the opposite approach, terms represented as vectors in a document space, is also sometimes used.

[15] Adapted from [PFTV92]

The following figure[16] illustrates the above theorem:

$$\begin{bmatrix} \\ \\ & \mathbf{A} & \\ \\ \\ \end{bmatrix} = \begin{bmatrix} \\ \\ & \mathbf{U} & \\ \\ \\ \end{bmatrix} \cdot \begin{bmatrix} \sigma_1 & & \\ & ... & \\ & & \sigma_N \end{bmatrix} \cdot \begin{bmatrix} \\ & \mathbf{V}^T & \\ \\ \end{bmatrix} \tag{3.2}$$

Since the singular vectors $\mathbf{U}$ and $\mathbf{V}$ and the singular values in $\mathbf{\Sigma}$ appear in sorted order (decreasing magnitude), the D first singular triplets (row from $\mathbf{U}$, singular value from $\mathbf{\Sigma}$, column from $\mathbf{V}^T$) multiplied together provide the best D-dimensional least square approximation $\mathbf{A}'$ of the original matrix $\mathbf{A}$. The first singular triplet thus captures the best 1-dimensional approximation to the original document/term vectors, the first and second thus capture the best 2-dimensional approximation to the original document/term vectors and so on. Only using the D leading singular triplets is known as *Reduced Singular Value Decomposition*. This technique of only using a limited number of the leading singular triplets thus represent the dimensional reduction that is at the core of LSA . D is known as the LSA-dimension and is a very important parameter in LSA. Reduced SVD is illustrated below:

$$\mathbf{A} \approx \mathbf{A}' = \mathbf{U}'\mathbf{\Sigma}'\mathbf{V}'^T \tag{3.3}$$

Where $\mathbf{U}'$ is a $M \times D$ matrix, $\mathbf{V}'$ a $N \times D$ matrix and $\mathbf{\Sigma}'$ is a $D \times D$ diagonal matrix. (Reduced) SVD thus computes 2 sets of orthogonal singular vectors $\mathbf{U}'$ and $\mathbf{V}'$ that form the basis of a new abstract subspace. This new abstract subspace is known as LSA-subspace and is where LSA perform its "magic". In LSA the left singular vectors $\mathbf{U}'$ provide a mapping from the original term space to the newly generated abstract LSA-subspace. Whereas the right singular vectors $\mathbf{V}'$ in turn provide a mapping from the original document space into the abstract LSA-subspace[17] [Ler99].

The important point is that LSA utilises (Reduced) SVD to perform an "optimal" dimensional reduction – on all the projections, on all the possible subspaces having the same number of dimensions (D), the LSA/SVD projection has the lowest least square difference to the original document vectors. So in that sense (Reduced) SVD is an optimal solution to dimension reduction [CTN03]. This "optimal" dimensional reduction is really the "gem" responsible for LSA's interesting properties, since it both reduces noise and at the same time uncovers latent information from the original term-document matrix. It works because the leading singular triplets capture the strongest and most meaningful regularities in the original data [Ler99].

The actual algorithms for computing the Reduced SVD (the D leading singular triplets) for a given sparse matrix are quite sophisticated and will not be given here, but usually run in $O(3Dz)$, where z is the number of non-zero elements in the original matrix (the number of document/term relations in the document collection). The traditional SVD algorithms (e.g.

[16] Adapted from [PFTV92]

[17] NB: In this text we have documents as columns and terms as rows, but LSA/SVD would work as well if the original matrix was transposed, having documents as rows and terms as columns.

Lanczos, Ritz-Raleigh iteration etc.) are global in the sense that they require the complete term-document matrix as input. However, Brand [Bra02] and [Bra03] have propose an incremental[18] SVD algorithm with $O(MND)$ complexity for calculating the D leading singular triplets for a $M \times N$ original matrix. This incremental approach might be useful in information retrieval applications where new documents are added to the collection on a regular basis.

3.2.3 Using LSA in Connection with Document Clustering

Due to the beneficial effects mentioned in the above section, LSA has been widely applied within document clustering research. The conventional[19] way of using LSA in clustering, is to project document vectors into the D-dimensional LSA-subspace (using the right set of singular vectors for the mapping) [Ler99]. In this way each document gets represented by a D-dimensional vector in LSA-subspace, where each dimension is an abstract linear combination of the actual terms. Traditional clustering algorithms (e.g. K-Means) can then be applied to do the actual clustering in this reduced dimensional subspace. The distance measure used in LSA-subspace is usually, but not always, the cosine between the document vectors. Experiments show that this measure works well, and according to [LFL98] there are some weak theoretical reasons for preferring it over other methods (e.g. the Euclidean distance measure).

Good results have been reported using this approach[20], which have two immediate benefits, the first being that the calculations involved in the clustering algorithm are scaled down significantly due to the reduced number of dimensions in LSA-subspace. Secondly many of the benefits discussed above – noise reduction, dampening of synonymy and polysemy and uncovering of latent information – improve the performance of most clustering algorithms significantly[21].

However, using LSA for document clustering is also connected with some inherent challenges. The first challenge is that there, given a collection of documents, are no commonly agreed upon way to determine the optimal dimension of the reduced LSA vector space[22]. If too small an LSA-dimension is chosen, important information patterns in the collection might be lost. In many cases the optimal dimension thus have to be empirically decided for a given document collection. However, Landauer and Dumais [LD97] have, based on numerous experiments within different LSA applications, found 300 dimensions to be a good compromise when dealing with large data sets.

Besides finding the optimal LSA-dimension, the effective usage of LSA require sophisticated fine tuning of various elements that influence the quality of the method. By some this tuning is almost viewed upon as an art. Some of the important factors (including LSA-dimension) that influence the quality of LSA are listed below[23]:

[18] "Incremental" means that the algorithm dynamically calculates the result as the documents are added one at a time.

[19] As shall become apparent soon, we have chosen to use our own novel approach to utilising the benefits of LSA in connection with document clustering.

[20] See [GEG03] and [Ler99]

[21] During the implementation, we have experimented with the traditional approach of clustering directly in the LSA subspace. However, we have found this approach much slower than the keyword-based approach, which we will introduce below.

[22] In [Ler99] a method for finding the optimal dimensionality is proposed, but we have not found any additional research verifying the correctness of this method.

[23] List adapted from [NPM01]

- Preprocessing (stop-word removal, stemming, weighting of terms etc.).
- Normalisation of document vectors.
- Choice of LSA-dimension.
- Choice of similarity measure for clustering.

Grönqvist [Gro] points out some other challenges related to LSA. Most notably he argues that since LSA works directly on the term-document matrix, it only looks at documents as "bags-of-words" and important information, such as the structure and word order of the documents, is hence lost. Grönqvist mentions usage of n-gram[24] terms as a possible solution to this and other shortcomings of LSA – an interesting idea that we will touch upon below in connection with the section on preprocessing.

3.3 Commonly Used Preprocessing Techniques

As mentioned in the beginning of this chapter, the above clustering methods depend on various preprocessing techniques to achieve optimal quality and performance. Selecting the best preprocessing methods for a given clustering algorithm is almost an art, but we will try to approach this from a scientific point of view and discuss some of the more commonly used preprocessing techniques in this section.

3.3.1 Stop Words and Other Kinds of Term Filtering

When indexing large quantities of text, it is generally agreed upon [BYRN99] that a lot of the words that are indexed are irrelevant for querying and other purposes (in our case, clustering). Different classes of such irrelevant words exist, and some of these are only irrelevant for some contexts. Filtering irrelevant words can reduce the size of the database as well as serving the (for us) critical purpose of removing noise from the database.

Nouns generally carry the most information, whereas other large word classes such as verbs, adverbs and adjectives, don't carry as much information. Some filtering approaches focus on dealing with this.

We regard term filtering as falling in two distinct classes: *Crawl-time Filtering* and *Post-crawl Filtering.*

Crawl-time Filtering

Crawl-time filtering includes all rules that can be set up before actually fetching the data. This includes stop-lists, numbers not falling into specific formats (e.g. maybe we want to keep dates) and 1-character words. Depending on the topic covered in the documents that are indexed, these can be modified and other rules may be added.

[24] N-grams are n-word (n usually being 2 or 3) sub-phrases that often occur in natural language, e.g. "Information Retrieval", "New York City" etc.

A stop-list is a list of *stop-words* – words that don't convey any practical (meaningful) information. This includes very commonly used words such as “a”, “be” and “few”. An example of a general stop-list is found in [FBY92, pp. 114].

When creating stop-lists or filters, it is important to keep the context that will be indexed in mind – what topics does this cover? Does it make sense to index the word “computer” if you are indexing the site of a computer science faculty?

Post-crawl Filtering

In post-crawl filtering, we can take a step back and view the overall database content. This enables us to identify stop word candidates by looking at their document frequencies. Words occurring in more than 80% [BYRN99] of the documents are very good stop-word candidates. Conversely, words only occurring in a few (maybe 1-2) of the documents in the database are very likely misspellings and shouldn't be indexed either.

However, care should be taken, when automating these operations – among the 200 most used words in English litterature are “life”, “time” and “war” [FBY92] – all words that shouldn't be removed. This problem can be solved using “white-lists” specifying which words should be included in any case.

Textual Contents

The downside of using the above filtering measures is that we might lose some of the implicit information from the text. Especially English does not concatenate conjoined words and thus some words or titles might be destroyed if for instance one part of the word is a stop-word and is not indexed.

An often seen practice is then to keep the words for some purposes, but mark them that they are not to be used alone. This will allow the user to search for complete phrases and will help by reducing the noise (better clustering) and enabling n-gram discovery (see section 3.3.4). The downside of this is the increased size of the database/index file.

For languages using only concatenated nouns such as Danish, it would be relatively straightforward to implement a dictionary with all verbs, adverbs etc. to mark all non-nouns, but since English has a different structure, advanced semantic parsing is required to mark these words. This we have deemed out of scope for this project.

3.3.2 Stemming

Different syntactic variations of a word generally describe the same concept. Present and past tenses of the verb *walk* (i.e. *walk*, *walks* and *walked*) and singular and plural forms of the noun *lamp* (i.e. *lamp* and *lamps*) are examples of words of different forms that for all information retrieval purposes would sit within the same context. Besides from reducing the dimensionality of the term space and thus making clustering easier, resolving these issues also reduces the total size of the index files, and thus the memory and storage requirements.

A number of different approaches to interpreting or discovering these variations exist, the process of combining the syntactic variations are known as *conflation* [FBY92]. Conflation can either be performed manually, for instance by defining regular expressions, or automatically. The automatic conflation algorithms are commonly known as *stemming algorithms*, which comes from the fact that we are looking for the common "stem" of words. Stemming is often performed in the preprocessing stage[25].

When performing automatic conflation, one must always consider the consequences of *overstemming*. Overstemming means to conflate two mostly unrelated words such as "engineer" and "engine". This will as a first consequence reduce information retrieval precision, when simply searching a database. The result of this is poorer information retrieval quality, but is rarely something that the user notices directly.

The more severe consequence for our purpose is that we will potentially be forming clusters of completely unrelated concepts, which in the case of a more-like-this system could yield results that don't resemble the original page at all. In our case, understemming (i.e. not conflating some word variations such as *organisation* and *organisations*) is thus preferable to overstemming.

Below, we will give a brief overview of the currently available conflation methods as outlined in Figure 3.5 (adapted from [FBY92]).

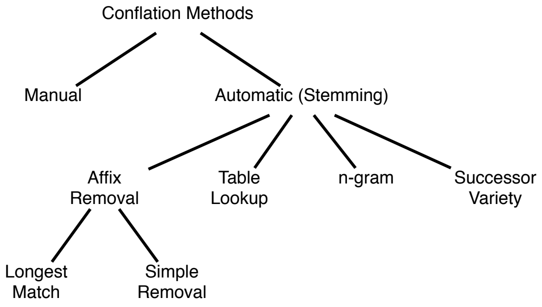


Figure 3.5: *Taxonomy of current conflation approaches.*

Manual Conflation As mentioned above, manual conflation simply covers a set of conflation rules based on observations of the term set. Since manual conflation quickly becomes infeasible as the term set or document set grows beyond human capacity, various automatic means become necessary.

Table Lookup Stemming using table lookup utilises a number of tables that contain the stem of the word and its different syntactic varieties. This can obviously be implemented to be very fast, but has a problematic limitation, namely that these tables are required. Such tables do not exist, or only exist for very limited contexts.

n-gram An often seen stemming algorithm is the shared digram method [FBY92] where all two-letter pairs (*digrams*) are extracted from the words in the corpus, those words having the

[25] In search engines, users' queries are also often stemmed.

most digrams in common according to some similarity measure are then made to be the same stem. The problem here is that we have to re-estimate or optimise the similarity requirements for different document corpora.

Successor Variety The successor variety of a string is the number of different characters that follow it in words in some body of text. By observing the successor varieties of all prefixes of the string, a steady decline or constant value for the successor varieties is expected until a sudden rise in the varieties will signify that a word stem has been determined. Consider for instance how the successor varieties would look for the word "reading" from a text body containing "read", "reads", "reading" and "rent". Obviously, the word "read" would be the stem of "reading".

This approach is very promising - especially since it could be applied to different languages without modification. However, the risk of overstemming plus the necessity to make the cut-off requirements flexible depending on the size of the document corpus limits the uses of this algorithm.

Affix Removal Stemmers using affix removal remove the suffix and/or prefix of words to produce the stem. The simplest affix removal stemmers simply truncate the words and simply keep e.g. 60% of the word or a set number of characters as the stem. This, of course, serves some purposes, but this approach should only be applied when working in small term spaces with very specific topics – otherwise the risk of overstemming and thus merging two or more unrelated concepts becomes to great.

The more intelligent approach to affix removal is called "Longest Match", where an iterative longest match stemmer removes the longest possible string of characters from a word according to a set of rules. This process is repeated until no further characters can be removed from the word. If the stemmer contains enough (correct) rules, most word variations can be safely conflated without the risk of overstemming.

Recoding Even though most words can be conflated using the Longest Match stemming method, one issue still stands out; vowel transformations. For instance "Ponies" may be stemmed to "Poni", but this will not be conflated with "pony" unless further transformations are applied. Such transformations are known as *recoding* and are also based on a (simpler) set of rules.

Porter Stemmer

According to [FBY92], affix stemming using the longest match approach is by far the most popular. Even though the other approaches to stemming show promise for different applications, our main concern is to have a stemming approach that will not require us to reoptimise our stemmer every time we choose to change to a different term collection.

Because of this, we have chosen to use a longest match affix removal stemmer called the Porter Stemmer, which is freely available[26]. The Porter Stemmer has the additional advantage of not only containing affix removal stemming rules, but also recoding rules.

[26] See `http://www.tartarus.org/~martin/PorterStemmer/`

3.3.3 Term Weighting

As discussed in section 2.1.2 about the vector model, it is an advantage if the the relation between terms and documents is not simply binary. In order to get a better approximation of the interrelationship between terms and documents, non-binary weights are often applied. However, it is far from given how to best calculate these weights. Various methods ranging from simply using the term frequencies to very complex statistical measures have been proposed. These methods are normally referred to as *weight functions* since they all result in a weight for each term-document relation.

The choice of weight function significantly influences the result of a clustering algorithm (and for that matter information retrieval in general). There is no commonly accepted theory detailing which kinds of weight functions are best suited for a given purpose. Instead, various empirical studies exist containing different and sometimes conflicting evidence for the performance and quality of different weight functions. Hence, we will briefly discuss some of the more commonly used weight functions[27] in this section.

According to [NPM01] it is often convenient to express the transformation done by weight functions as a product of two numbers - a local and a global weight (function).

$$W_{t,d} = L(t,d) * G(t) \tag{3.4}$$

The local weight function $L(t,d)$ represents the weight of the term t in document d and is a measure of how related the given term and document are. The global weight function, on the other hand, represents the weight of a term in the entire document collection, and is a measure of how important the given term is as a descriptor of documents in the given collection. The purpose of a good global weight function is to weight terms to minimise intra-cluster distances and maximise inter-cluster distances. This is often done by increasing the weight of the terms that best distinguish the documents containing them (e.g. terms that are relatively rare in the collection).

The most commonly used local weight function is simply the term frequency, $tf_{t,d}$, which is the frequency of term t in document d. This function works quite well, since words central to a document's content and theme often occur with high frequency. However, there is room for improvement and more elaborate schemes that use information about the term's position within the document to calculate a term weight $tw_{t,d}$ are sometimes also used. Here, terms appearing within titles, or otherwise emphasised have higher weights than terms that only occur in the main text. Usually term frequencies still play an important role in such weighting schemes, albeit reduced or emphasised by the other factors.

As for global weight functions, many different approaches exist. The most simple global weight function is the unity function:

$$G(t) = 1 \tag{3.5}$$

This function corresponds to simply using the local weight function without any global weighting, and is often used as a baseline for comparing whether global weighting improves performance for a given application.

[27] Based on the elaborate discussion on this in [NPM01].

Normalisation

Normalisation of the document vectors is another commonly used global weight function. Various ways of normalising the document vectors exist. One popular method is given in [BYRN99, pp. 29]:

$$G(t) = \frac{1}{max_i(L(i,d)} \tag{3.6}$$

This method simply takes the inverse of the maximum term weight in the given document (vector). Doing so, the method ensures that the weights assigned to documents of different sizes are scaled to comparable magnitudes. Another common normalisation approach is given in [NPM01]:

$$G(t) = \frac{1}{\sqrt{\sum_i L(i,d)^2}} \tag{3.7}$$

Here, the method uses the standard normalisation technique of dividing the document vector with its length. It is thus ensured that all document vectors have unit length and their position in the term space hence only depends on the textual content of the documents and not their lengths.

IDF

In connection with clustering, the global weight function known as *IDF* is often applied. IDF is an abbreviation for *Inverse Document Frequency* and is calculated as follows:

$$G(t) = \frac{N}{df_t} \tag{3.8}$$

Or, alternatively as *log-IDF*:

$$G(t) = 1 + log(\frac{N}{df_t}) \tag{3.9}$$

df_t is the document frequency of the term t, i.e. the number documents where the term t occurs. N is simply the number of documents in the collection. This weight function increases the weight of rare terms since it is believed that these are better to distinguish document clusters.

Entropy-based Methods

Another class of global weight methods are based on the information-theoretic notion of entropy. The entropy is calculated using the conditional probability of the document d under the condition that the term t occurs (i.e. the probability that a randomly selected occurrence of term t is in document d):

$$p_{t,d} = \frac{tf_{t,d}}{gf_t} \tag{3.10}$$

Where gf_t is the global frequency of the term t, i.e. the number of times the term t occurs in the collection[28]. The entropy of term t is then given as:

$$E(t) = \sum_i (p_{t,i} * log(p_{t,i})) \tag{3.11}$$

The first global weight function based on entropy is simply the absolute value of the entropy (the entropy is always a negative number or 0):

$$G(t) = -E(t) \tag{3.12}$$

The entropy measures the uniformity of the term's frequency in the documents where it occurs – i.e. the more uniform the distribution, the higher the global weight. The rationale behind using the entropy for a global weight function is that using the term in a consistent manner in a small group of documents is an indication that the term might relate these documents in a "good" cluster. However, the document frequency is not factored into the equation, which might present a problem in connection with terms that occur more or less uniformly in the entire collection (e.g. common navigation elements and headers on webpages).

A global weight function that compensates for this, while still using the interesting properties of the entropy is *log-entropy*:

$$G(t) = 1 + \frac{E(t)}{log(N)} \tag{3.13}$$

In this scheme, the more uniform the occurrence of a term (in the entire collection), the less interesting it is seen from a clustering perspective and is thus assigned a lower global weight.

[BN04] explains log-entropy in the following way:

> "... [log-]entropy gives a measure [of] how well a word is suited to separate documents by keyword search. For instance, words that occur in many documents will have low [log-]entropy. The [log-]entropy can be seen as a measure of the importance of a word in the given domain context. As index words a number of words that have a high [log-]entropy relative to their overall frequency have been chosen, i.e. of words occurring equally often those with the higher [log-]entropy can be preferred. Empirically this procedure has been found to yield a set of relevant words that are suited to serve as index terms."

[28] Global frequency is sometimes also referred to as collection frequency (cf_t).

Final Remarks

[BYRN99, chap. 2] recommends a global weight function based on both normalisation and log-IDF – effectively multiplying these two functions to form a new global weight function. However, [Dum93] recommends using log-entropy in connection with LSA[29]. [NPM01] on the other hand found some evidence that "pure" entropy performed slightly better than log-entropy also in connection with LSA, but they mention that this might be specific to the document collection they used.

3.3.4 Usage of N-Grams

Most clustering methods only consider documents as "bags-of-words" without taking word order and the textual structure of the document into consideration. To preserve some of this information, n-gram terms are sometimes extracted and used.

An n-gram is an n-word (n usually being 2 or 3) sub-phrase often occurring in natural language. These sub-phrases often have a meaning that is somewhat different from the meanings of the constituent words. This could be sub-phrases such as "keyword extraction" and "United States of America".

As mentioned, n-grams are particularly important in connection with English, since English contains many n-word compounds (e.g. flower shop) as opposed to most other Germanic languages as well as Latin languages, where most compound words are formed by concatenation – i.e. solid compounds ("flower shop" in Danish becomes "blomsterbutik").

In connection with clustering, n-grams often make better conceptual descriptors compared with the single word terms, since they capture more of the implicit information of the document. [Hul03] found some evidence that n-grams improve the quality of automatic keyword extraction, and make the keywords closer to a manually assigned set made by experts. In addition, [Gro] suggests using n-grams in connection with *Latent Semantic Analysis* to remedy some of its weaknesses (see section 3.2).

Obviously, we expect diminishing returns with regard to the length of n-grams when using n-grams in connection with clustering. Many good bigrams exist in most documents, whereas fewer common trigrams, quadgrams and beyond exist in typical document collections. Since, most methods of extracting n-grams are often computationally demanding, it makes sense to only focus on bigrams, and perhaps also trigrams.

One class of approaches to n-gram extraction employs various kinds of statistics to find relevant n-grams. Alternatively a list of known/common n-grams can be used. However, it is often difficult to cover all domains with one general list. Using domain-dependent lists is thus often required to locate relevant n-grams with this method.

[29] See section 3.2.

3.4 Postprocessing

As already mentioned in section 2.3.2, we have chosen to focus our efforts on two applications of document clustering; finding similar documents (more-like-this) and search result clustering.

3.4.1 Finding Similar Documents (More-Like-This)

Finding similar documents is a popular way of enhancing the search experience, allowing the user to easier identify pages containing desired information. This is often implemented as a "similar pages" link next to each search result. If the user for instance searches for "jaguar" with the intent to find pages concerning information about the vehicle, he might be presented with results both about vehicles and animals. If he clicks on a similar page link besides one of the results regarding vehicles, the more-like-this system should only present pages concerning vehicles.

Another scenario for using more-like-this is to display links to pages resembling the current page in a small additional frame while browsing a website.

However, it is not always clear what constitutes "similar pages", since the definition can be different to different people – for some people, more-like-this means pages in the same category and for other people more-like-this means pages containing information on exactly the same topic. We have chosen to use the following definition:

> "Similar pages are pages that fall within the same category and hence contain (more-like-this) information on related topics."

The more specific a given page is, the more difficult it becomes to find related pages within the same category and a choice should be made whether to display less relevant pages or to display less pages. Tables of similar page information can be predetermined offline for maximum performance at run-time.

3.4.2 Search Result Clustering

Today, most search engines operate by returning a long (ranked) list of results matching the user's query. However, as already mentioned, the cluster hypothesis states that similar documents tend to be relevant to the same requests. It is thus often advantageous to not only rank search results, but also to categorise them according to the main topic of the pages. However, the process of manually categorising the pages of a website is often tedious and expensive. Document clustering has thus often been used to automatically categorise a search result into topic groups (clusters).

This can either be done as a flat clustering of the search results, where the result is divided into a number of hard or soft clusters. Another approach uses a hierarchy to capture the subsumption relationship between different topics (e.g. "software engineering" is a subtopic of "engineering"). The user is thus presented with a topic hierarchy that he/she can navigate through to find the pages that are relevant.

As an alternative to dividing the search result into clusters, clusters could also be presented as part of the search result (in a Yahoo-like fashion). The user is thus presented with categories that might interest him/her and these categories might contain pages that would not be found with the original query.

Due to the nature of search results (the number of possible queries and corresponding results is enormous), search result clustering has to be performed online (on-the-fly). However, this system can of course still utilise an underlying offline clustering of the whole document collection to generate clusters relevant to the given search results.

Chapter 4

Chosen Approach

This chapter starts by outlining and motivating our chosen two-stage approach to document clustering. We then motivate our choice of clustering algorithms, before moving on to a discussion of the architecture we have designed for the clustering toolkit.

4.1 Using Core-Terms/Keywords to Represent Documents

We have decided to split the *document clustering* stage (stage three in figure 3.1 on page 17) into two distinct stages:

1. **Keyword Extraction** In this stage, keyword or core-terms are assigned to each document based on its main theme/topic.
2. **Keyword-based Document Clustering** In this stage, the assigned keywords are used as features for clustering using one of the selected clustering algorithms (see below).

The two-stage approach is inspired by the experiments by Schütze and Silverstein in [SS97], discussed earlier, which showed no loss of quality, but a drastic speed-up in connection with truncation according to term weight. We expect that better keyword extraction schemes than truncation will yield even better results with regard to quality.

We have thus essentially chosen to utilise keyword-based clustering as defined in the precious chapter, because we believe that this method carries some interesting advantages and can be used with most of the approaches to clustering outlined in section 3.1. The clustering algorithms are simply used with keywords as features instead of the full term space. We consider this approach natural, since a document's content is often best described by a few well-chosen keywords rather than by the full set of words – short descriptions are often the best descriptions since they are long enough to capture the essence, but leave out the less relevant information (and noise). Structuring and organising information has always required eliciting the essential information in one way or another. This is for instance utilised in connection with the abstracts and keywords that are often used in academic papers.

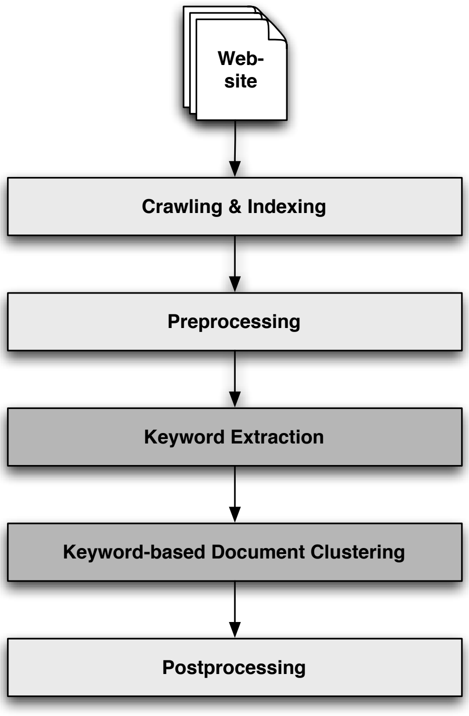


Figure 4.1: *We have split the Document Clustering stage into 2 stages: Keyword Extraction and Keyword-based Document Clustering.*

Furthermore, this approach allows for interesting algorithms such as frequent itemsets and lattice-based algorithms that require a limited set of features with large overlap. In connection with more conventional clustering approaches, the use of keywords instead of the full term space results in a huge performance advantage as already touched upon. Since Mondosoft is interested in using clustering technologies for clustering fairly large websites (100,000+ pages), this performance gain is necessary to make clustering feasible.

The method of representing documents by a limited number of keywords is obviously connected with a loss of information. However, we believe that, if keyword extraction is done correctly (i.e. by choosing "good" keywords), most relevant information will be preserved, while noise and irrelevant information are filtered out. This, of course, requires research into good keyword extraction methods that are able to provide the necessary quality of keywords as well as ensuring the keyword overlap necessary for clustering.

4.2 Chosen Algorithms

We have attempted to choose a set of clustering algorithms that cover most of the major classes mentioned in the previous chapter. However, some classes of algorithms are not particularly interesting with regard to the scope of this project.

Partition-based algorithms are very common and often offer a good trade-off between speed and cluster quality. However, since we prefer a hierarchical clustering, we have chosen to use a partition-based algorithm as basis for a divisive hierarchical clustering. For this purpose, we have chosen K-Means, since it is probably the most well-known clustering algorithm around, and it will thus serve as a baseline to compare other algorithms against. The divisive version of K-Means we have chosen to use, is often referred to as *Bisecting K-Means*.

Hierarchical clustering methods readily offer the hierarchical clustering structure, we desire, and we have therefore chosen to implement one algorithm from each of the two main branches of the category; CURE and PDDP. We have chosen CURE to represent the class of agglomerative hierarchical algorithms, since it is fast compared to other algorithms in this category and since it has an interesting way of handling clusters of non-convex shapes. PDDP has been chosen to represent the class of divisive hierarchical algorithms, since it relies on an interesting approach that promises excellent speed combined with good quality clusters.

As already mentioned, our chosen approach relies on keyword-based clustering. This allows us to implement two lattice-based clustering algorithms: GALOIS and lattice-clustering based on Apriori. GALOIS is an incremental approach to building a concept lattice from a set of documents that we believe represents one of the fastest approaches to building such a lattice. However, realising the close connection between frequent itemsets and concept lattices, we have also chosen to implement an extended version of Apriori. This version is able to produce "pruned" concept lattices that only contain clusters/concepts that have a minimum support in the document collection. Since Apriori was developed to mine frequent itemsets in databases of purchasing transactions, we hope that this way of generating concept lattices will scale well and be significant faster than GALOIS.

As for model-based clustering algorithms, we have chosen to focus our efforts on *Latent Semantic Analysis*, since we hope to benefit from some of the promising properties of this approach to dimensional reduction. However, instead of using LSA directly in connection with clustering, we will utilise it to extract keywords fulfilling the criteria set above. With regard to neural network-based clustering methods, our literature study shows they either require training (supervised methods) or perform too slowly to be feasible in connection with the sizes of the document collections that we would like to be able to cluster.

The other mentioned methods in the analysis were shown to be either irrelevant or infeasible in connection with the scope of this project.

We have thus chosen to implement the following algorithms:

- Keyword Extraction based on LSA[1]
- Bisecting K-Means
- CURE
- PDDP
- GALOIS
- The Apriori-based lattice clustering scheme

[1] We will also implement a keyword extraction scheme based on pure truncation (inspired by the positive results in [SS97]) that can be used as a baseline to compare the LSA method against.

4.3 Toolkit Architecture

In this section, we provide an overview of the architecture that we have designed for the toolkit. The architecture is somewhat idealised, the actual implementation varies a bit for different practical reasons.

During the design phase, our main emphasis has been on setting up an architecture that makes experiments with (and evaluation of) preprocessing methods, keyword extraction, clustering methods and postprocessing methods easy.

4.3.1 A Modular Design

We have chosen to design a modular architecture that allows for ease of customisation and integration of different algorithms in the different stages of the clustering process. Using a modular approach also makes it possible to expand the "repertoire" of algorithms in the different stages, if so desired.

This modular architecture requires that interfaces between the different stages are defined in a flexible way, in order to allow different combinations of algorithms to operate together (i.e. to make sure that the output of one stage, regardless of the used algorithm, is compatible with the input of the next). Furthermore, based on a given configuration file, the modules should be interchangeable at run-time, allowing on-the-fly switching of algorithms without the need to recompile.

Figure 4.2 below shows the different classes of modules in our architecture, which will be described further in the following sections. As is seen, the different classes of modules are an extension of the process flow in figure 4.1, omitting the crawling and indexing stage, which is handled by MondoSearch™.

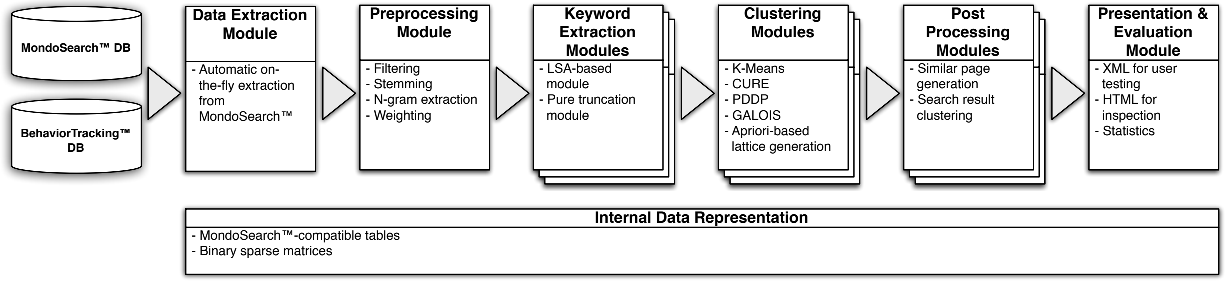


Figure 4.2: *The different classes of modules in the architecture.*

Stage 1: Data Extraction

This stage represents the primary interface into the MondoSearch™ databases (including the necessary BehaviorTracking™ data). The module is responsible for extracting the term-document matrix on-the-fly, and should also handle other information requests to the databases from subsequent stages.

The primary output of this stage is a sparse feature-document matrix with terms as features. Additionally, conversion tables for rows and columns into terms and documents are also produced.

Stage 2: Preprocessing and Weighting

In the second stage, the term-document matrix produced in stage 1 will be transformed through the chosen preprocessing steps, including:

- Stop-word filtering and other kinds of post-crawl filtering[2]
- Stemming
- N-gram extraction
- Local and global weighting[3]

The primary output of this stage is a new sparse feature-document matrix with word stems or n-grams as (document) features. Additionally, conversion tables for rows and columns into n-grams/word stems and documents are also produced.

Stage 3: Extraction of Keywords

This stage is responsible for assigning a number of keywords/core-terms to every document based on the feature-document matrix output by stage 2.

The primary output of this stage is a keyword table containing the keywords and corresponding weights for each document.

Stage 4: Document Clustering

This is of course the core stage concerned with the actual document clustering based on the keywords stored in stage 3. Modules in this stage contain the implementations of the chosen algorithms that should work directly on the keyword table.

The primary output for the hierarchy-based algorithms (K-Means, PDDP and CURE), is a table specifying each cluster and the documents contained in it. For the lattice-based algorithms (GALOIS and extended Apriori), the primary output is a table specifying the lattice's nodes and their linking structure.

In addition, both types of algorithms also output an inverse table containing document-to-cluster/node relations to speed up the subsequent postprocessing stage.

[2] For practical reasons, our actual implementation performs this directly during data extraction for maximum efficiency.

[3] Depending on which weighting algorithms, the user has chosen.

Stage 5: Postprocessing

This stage is primarily concerned with finding similar pages and, to a lesser degree, search result clustering as specified in the scope of the project.

Modules in this stage contain functions for building more-like-this tables for either a hierarchical clustering or a lattice-based clustering. Time allowing, the module should also contain implementations of a search result clustering for both classes of clustering algorithms[4].

The primary output of this stage is a table containing references to similar pages for each document. The format of this table should be independent of the clustering algorithm applied in stage 4. Search result clustering is as mentioned an online process, and we do not need to generate any static output during the offline clustering process.

Stage 6: Presentation and Evaluation

This stage consists of various utility functions that output information/data that can be used to evaluate the previous stages. Output functions should include:

- Functions to generate HTML versions of the keyword table and the similar page table, allowing these to be inspected by human users.
- Functions to generate XML versions of the similar page table, allowing it to be used as the basis of a user test of the similar page feature for the different algorithms.
- A simple search functionality that is able to output the result clustering of 1-word searches. This should act as a preliminary "mock-up", allowing us to determine the feasibility of search result clustering.
- Functions to generate statistics for a given keyword table and to compare two keyword tables to determine the percentage of changed keywords (for instance due to a change of algorithm or algorithm parameters).

4.3.2 Internal Data Representation

The interfaces outlined above use two primary data structures; tables and sparse matrices. For table-based data structures, we have chosen to use the same kinds of tables as those used by MondoSearch™ to make later integration into MondoSearch™ easier. Doing this we also save the work involved in developing a table-based database ourselves. For matrices, we have chosen to use a standard binary sparse matrix format.

[4] Search result clustering for hierarchical algorithms has not been implemented due to time constraints.

Chapter 5

Implemented Preprocessing

In this chapter, we will outline the different preprocessing substages that we have implemented. First, we will outline the extraction of data from MondoSearch™. Then we move on to describing the implemented post-crawl filtering of terms. After this, we will discuss the implemented stemming scheme.

Following this, we will briefly outline the implemented local and global term weighting functions. Finally, we will detail the two schemes we have designed to extract relevant bigrams that can be used as document features.

5.1 Extraction of Data from MondoSearch™

We have implemented a data extraction module that is directly integrated with MondoSearch™ in the sense that it uses same the database files that the "live" version of MondoSearch™ queries. It is thus very straight-forward to perform clustering on websites crawled and indexed by MondoSearch™. Being able to take advantage of the many hours that have been used building the MondoSearch™ crawling and indexing system gives us a clear advantage compared with other researchers in the field of information retrieval. In this way we have great freedom in the choice of test data to use when optimising and evaluating our algorithms.

The main purpose of the data extraction module is build a term-document matrix that can be used by the subsequent preprocessing stage. Our implementation allows term-document matrices with either document frequencies or MondoSearch™ weighting[1]. To save space, the term-document matrix is stored in sparse form only saving non-zero values. In practice, we use a standard binary sparse matrix format[2] to store the compiled matrix on disk. As a complement to the matrix, the extraction module also builds conversion tables that are used to translate between row and column numbers and the corresponding pageIDs and wordIDs that are used internally in the MondoSearch™ database.

In addition to generating the term-document matrix, the data extraction module also handles extracting the following information from the MondoSearch™ databases:

[1] See section 5.4 below, briefly introducing the Mondosearch™ term weighting scheme.

[2] The definition for the file format can be inspected at: `http://tedlab.mit.edu/~dr/SVDLIBC/SVD_F_SB.html`

- Lookup of actual words (as strings) from wordIDs and vice versa.
- Lookup of page titles and URLs (as strings) from pageIDs.
- Determining the support of candidate bigrams in a given database.
- Getting the result of simple 1-word searches that can be used to test search result clustering.

5.2 Filtering of Terms

We make use of both *crawl-time* and *post-crawl* stop-word filtering methods. MondoSearch™ has its own built-in stop-list for every supported language and automatically filters/removes each stop-word. In addition to this, we have implemented an optional upper threshold (in %) on terms' document frequencies, thus essentially implementing post-crawl discovery and removal of stop-words. This was done because we found that webpages often contain header and navigation terms that are present on all (or most) pages on the given site. These words are normally not suited as keywords since they might be assigned to too many pages. However, on certain sites the upper threshold should be disabled since actual important words might also be part of header or navigation terms.

Besides from stop-word filtering, we have also implemented a few other (post-crawl) filters to improve quality and performance of our algorithms:

- A lower threshold (in number of documents) on terms' document frequencies. This is primarily done to improve speed and memory consumption, since this filter reduces the number of terms by an order of magnitude (see section 10.2.1). Such rare words are often spelling mistakes and are seldom very good for clustering (i.e. *noise*), since they are only able to form clusters that in most cases are too small to be relevant. In connection with our chosen keyword clustering based approach this is even more true, since rare terms decrease keyword overlap and consequently lead to documents not having common keywords for clustering.
- 1-character terms are also removed since these terms do not carry much meaning (if any) and thus make poor keywords.
- Finally, we filter out numbers with less than 4 digits since these rarely carry much meaning. However, numbers such as years and postal codes are sometimes important. We have thus chosen to accept numbers with 4 or more digits in order to preserve years and (Danish) postal codes, but we leave out short numbers that carry less meaning.

Our own informal tests on many different sites have shown that most of the above filters produce good results. However, it is important to point out that all implemented post-crawl filters are optional and it is thus up to the user to optimise the mix and settings of the filters to the given site.

5.3 Stemming

In this section, we will describe how we use the Porter Stemmer to create the database of features that we will base our clustering algorithms on.

5.3.1 Porter Stemmer

In section 3.3.2 we decided to use the Porter Stemmer for the term conflation operations that we need. In this section, we will describe how this is implemented in our system as well as the data representation for the conflated terms.

The Porter Stemmer is a longest match affix removal stemmer for English text. The Porter Stemmer has several sets of rules that are executed in turn to ensure maximal stemming and minimal overstemming. The Porter Stemmer is freely available on `http://www.tartarus.org/~martin/PorterStemmer/` where the original author maintains versions for different programming languages.

The Porter Stemmer does not offer any facilities for building up a database of conflated words, so in order to process our database, we have created a novel wrapping function that calls the Porter Stemmer to get the word stem and builds a *trie* containing information about the stemmed words. The wrapper takes the string representing the word, the word ID, the document frequency for the word and the collection frequency for the word.

5.3.2 Data Structures: Trie

A *trie* is a tree structure that stores strings in such a way that there is one node for every common prefix[3]. E.g. if the words “ape” and “apple” are put in the trie, they will be stored in branches coming from the common branch “ap”. Also see figure 5.1 for an illustration of this concept.

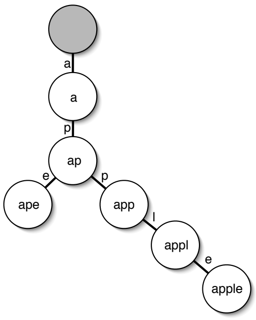


Figure 5.1: *A simple trie.*

The reason for using a trie instead of tables of some sort is that it is both efficient in terms of computational complexity[4], as well as in terms of memory usage, since we eliminate redundancy.

[3] See `http://www.nist.gov/dads/HTML/trie.html`

[4] $O(log_{26}n)$ for search and insert operations, where n is the number of characters in the word.

In a node in the trie, we store the word IDs of words that stem to this node, the accumulated frequencies from these words, the word ID of a *candidate word* and the frequencies of the candidate word. The reason that we store a candidate word instead of just using the stem, is that the word stems tend to be “less than a word” and thus are not very readable. Storing a candidate word gives us more readable keywords and lets us label clusters easier. The candidate word is simply the word with the largest collection frequency, which is usually the most general syntactic variation of the word. We have left the option of using document frequency instead of collection frequency, but we have seen the best results using the collection frequency.

Since the Porter Stemmer is an English-only stemmer, strings containing numbers and words containing non-English characters cannot be stemmed. Therefore, we have extended the trie to have an additional node in the top-most level simply containing a list of all non-english words and their weights. The final version of our trie is shown in figure 5.2.

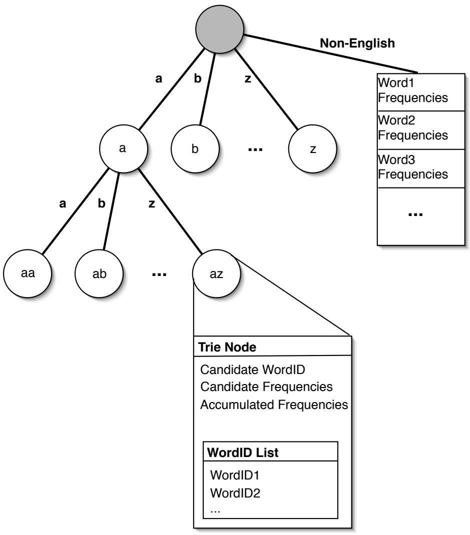


Figure 5.2: *The implemented trie structure for creating word stems.*

5.3.3 Data Structures: Stem Group

Storing the trie in a flat database proves impractical, so in order to store the stemmed words in the database, we have created so-called stem groups (see figure 5.3) that record lists of the conflated words as well as the candidate words.

As we mentioned above, both the accumulated document frequency and the accumulated collection frequency are stored for customisability - different term weighting techniques use different frequencies.

5.3.4 Modifications to the Porter Stemmer

When testing our stemming module, we discovered that some of the stemming rules resulted in overstemming. More specifically, we saw “universe” and “university” conflated and “engine”

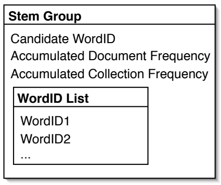

Stem Group
Candidate WordID
Accumulated Document Frequency
Accumulated Collection Frequency
WordID List
WordID1
WordID2
...

Figure 5.3: *Data representation for a group of conflated words.*

and “engineer”. We have modified the Porter Stemmer to avoid overstemming in these cases, and have not seen any cases of understemming resulting from our changes.

5.4 Term Weighting

As discussed in section 3.3.3, it is advantageous to split the weighting scheme into a local and a global part. Below, we detail our implementation of these two parts.

5.4.1 Local Term Weighting

We have implemented two different schemes for local weighting. The first scheme is simply based on using term frequencies. This scheme is commonly used and makes sense in most cases.

However, we were in the fortunate situation that Mondosoft has implemented a more elaborate weighting scheme in MondoSearch™ that is able to give better estimates of a term’s importance within a given document. The details of this scheme are confidential, but the main factors that influence weighting are given in [Mon02]:

> “... factors include the frequency of the word within a document (count); as well as the context the words are placed in (bold, font size, title, description, keywords, header tags [H1,H2,H3]) (weight); The frequency of the word within a document in comparison to the total number of words within the document (density); the use of keyword meta tags and titles; and the use of “rank word” and “rank document” meta tags, which are specific tags for use with MondoSearch...”

5.4.2 Global Term Weighting

Since different empirical studies show different global weight functions as optimal for clustering, we have chosen to implement the most common ones.

We have thus implemented global weight functions based on the following methods[5]:

- The *unit* function (used as baseline).

[5] See section 3.3.3 for a discussion of the different methods

- The *IDF* function.
- The *log-IDF* function.
- The *entropy* function[6].
- The *log-entropy* function.

With regard to *normalisation*, we have chosen to normalise the weighted document vectors before performing SVD on them. However, this is done as part of our keyword extraction and is therefore detailed in section 6.1.

5.5 Bigram Extraction

As we mentioned in section 3.3.4, using n-grams is a way of improving quality of clustering algorithms by remedying the common problem of clustering algorithms only considering documents as “bags-of-words”. We have invented two novel approaches for n-gram extraction. However, due to the limited time frame of this project, we have for now only implemented extraction of bigrams. It is important to emphasise that both of the below methods could be extended to also find trigrams or longer n-grams.

Both methods work directly on the term-document matrix and thus bypass post-crawl stopwords, other kinds of filtering and stemming. This is necessary since a constituent word in a relevant bigram might in itself be irrelevant. Furthermore, bigrams should not be stemmed since the bigram has a meaning in itself, not dependent of the constituent words.

5.5.1 Scheme 1: Using Behaviour Tracking Data

The first extraction scheme relies on usage statistics from Mondosoft’s behaviour tracking system. This method is thus somewhat specific to Mondosoft since other applications of clustering might not be linked/combined with a search engine.

The scheme simply consists of extracting n-grams by identifying n-word queries performed by users with a given minimum support (number of times the query has been performed). Each of these candidate n-grams are then checked against the document collection and are only kept if they have a given minimum support within the collection (either in terms of document frequency or in terms of collection frequency).

The advantage of this approach is that the extracted n-grams are both domain-specific and at the same time encompass n-grams that the users of the website find relevant. Our informal testing shows that this method yields good results (see appendix G.1 for a demonstration of this using the Harry Potter fansite MuggleNet[7]).

[6] In connection with entropy and log-entropy, the conditional probability, $p_{t,d}$ is calculated as $\frac{L(t,d)}{\sum_i L(t,i)}$ to accommodate for local weighting schemes other than term frequency. The distribution that we find the entropy of is hence not necessarily the frequency distribution, but the “importance distribution” of the term in the document collection.

[7] `http://www.mugglenet.com/`

5.5.2 Scheme 2: Using Frequent 2-itemsets

The second scheme relies on feedback from our lattice-based clustering methods to find keywords that frequently occur together. These n-gram candidates are compared against the collection as above to find which candidates have the required minimum support in the collection. In this way, we utilise a kind of "feedback loop" where the results of the previous clustering are used to extract bigrams for the current clustering. This is especially useful in connection with websites, since we envision that the clustering is updated regularly (e.g. during the weekly crawl).

This method is somewhat more "brute-force", but has the advantage of not requiring access to behaviour tracking data on users' search queries. If such information is not available, this method is the only current option. This method yields acceptable results, but the quality of the found n-grams is somewhat lower than with the above method. This is because some of the bigrams are simply words frequently occurring together, but not necessarily relevant to the users. However, since we only look at keywords, most of the extracted bigrams are still relevant and quite good (see appendix G.2 for a demonstration of this using a 10,000-page subset of Wikipedia).

Chapter 6

Implemented Keyword Extraction Algorithms

In this chapter, we will primarily discuss the LSA-based approach to keyword extraction that we have designed ("Synergy"). At the end of the chapter, we will also briefly discuss a simpler keyword extraction scheme based on truncation, which we have implemented as a baseline reference to compare Synergy against.

6.1 Synergy - LSA Based Keyword Extraction

6.1.1 Description

The two-stage clustering approach outlined earlier relies heavily on a good keyword extraction scheme. Good keywords that capture the main topic(s) of a given document are essential for the success of the subsequent clustering stage. We have thus spent a considerable amount of time researching different methods of keyword extraction.

However, we gradually realised that traditional keyword extraction schemes were not necessarily designed to find keywords that are good for doing subsequent clustering. In this connection, it is important to emphasise that the keywords we intend to extract are not merely keywords in the ordinary sense. They should rather be understood as a form of index terms, document categories or "core-terms" for the given document. But we use the term "keywords" since we still think that this is the notion that comes closest in describing the terms we want to extract – terms that both describe the essence/main topic of the document and at the same time are general enough to ensure the overlap with keywords of other documents, necessary for clustering. Our goal is to describe the entire document collection by a relatively small set of good keywords that links related documents together and unlink unrelated documents.

Traditional keyword extraction techniques might thus not suffice in extracting the kinds of keywords that we believe are necessary for clustering purposes. As a result we turned our attention to one of the more promising technologies that we have encountered doing our research into clustering techniques: *Latent Semantic Analysis* (LSA)[1]. We wanted to somehow harness

[1] See section 3.2.

LSA's interesting properties with regard to noise reduction, inference capabilities and dampening of synonymity and polysemy to extract good keywords matching the above requirements.

Keyword extraction is in fact often mentioned as a possible application of LSA, however, we have not, despite extensive research, been able to locate any references detailing how this can be done. So inspired by the different papers detailing LSA's properties (especially [LFL98]) we have invented our own novel[2] approach to keyword extraction using LSA.

The main idea behind our scheme is to utilise LSA to estimate the distance between terms and documents based on a reconstruction ($\mathbf{A}'$) of the original term-document matrix ($\mathbf{A}$) from the chosen number of leading singular triplets (the LSA-dimension D). The method exploits that SVD/LSA represents each document and term via the D abstract dimensions. Combining (by vector arithmetic) the vectors of a given term and a given document in $\mathbf{A}'$, the "closeness" between the document and the term is estimated from the abstract LSA-subspace [LFL98]. This estimate is a good measure of how semantically close a given term and a given document are. It has several interesting properties, that we take advantage of in our keyword extractor:

- Terms, not occurring in a given document, that are related to other terms in the document, will be estimated to be close to the document. Keywords not present in the given document can thus be inferred from other documents.

- Terms, occurring in a given document, that do not occur in any of the other documents that share many of the same terms with the original document, will be estimated to be less close to the document. "Noise terms" are thus dampened and less likely to become keywords.

- Terms that only occur in very few documents are also dampened. Keyword overlap is thus increased.

In this way, global information from the entire collection is used to improve the keywords extracted from a given document and to achieve the keyword overlap required for subsequent clustering. The degree to which the above properties come into effect can be controlled through the value of D (LSA-dimension). The fewer singular triplets (lower LSA-dimension D) the more the above effects kick in, and the extracted keywords will as a result become more general and less specific to the given document.

What's in a name?

We have named our keyword extraction scheme "Synergy", since it, as explained above. relies on global information from the entire document collection, when assigning keywords to a particular document. The documents considered together, thus possess more information than the mere sum of the information present in the individual documents – considered together, the documents form a synergy.

[2] We are not aware of other research using this exact technique for keyword extraction, however, we can of course not rule out that such research might exist.

Normalisation

As can be seen in the below outline of our keyword extraction algorithm, we normalise the term-document matrix (**A**) before we perform (reduced) SVD on it. This is done since the document vectors should be independent of the documents' lengths to avoid that large documents dominate the result. The distances between documents in term space should reflect semantic differences in content and not differences in the lengths of the documents.

Originally we had not included the normalisation step in the algorithm, but our testing showed that the keywords assigned to short documents often became very general (too much inference). We realised that this was due to lack of normalisation – the higher term weights present in large documents suppressed the lower term weights present in short documents when $\mathbf{A}'$ was constructed from the leading singular triplets. As a result we chose to normalise the document vectors, by dividing them with their length.

6.1.2 Algorithm

Input:

- A $M \times N$ term-document sparse matrix **A** (terms as rows and documents as columns) with z non-zero elements.
- The desired number of keywords per document k.
- The desired number of leading singular triplets to retrieve D (LSA-dimension).

Step 1: Normalise the document vectors (columns) of **A** by dividing the values of each document vector with its length $|a_j|$:

$$|a_j| = \sqrt{\sum_{i=0}^{M} a_{i,j}^2} \tag{6.1}$$

j refers to the current column vector (document).

Step 2: Perform (reduced) SVD on sparse matrix **A**, requesting the D leading singular triplets:

$$\mathbf{A} \approx \mathbf{U}'\mathbf{\Sigma}'\mathbf{V}'^T \tag{6.2}$$

Step 3: Calculate the approximation ($\mathbf{A}'$) to **A**, one column at a time, by multiplying the returned leading singular triplets together. For each column (document vector), only save the row number and the value of the k largest values. These row number/value pairs represent the found keywords and their "closeness" to the given document (weight).

$$\mathbf{A} \approx \mathbf{A}' = \mathbf{U}'\mathbf{\Sigma}'\mathbf{V}'^T \tag{6.3}$$

The k largest values are found by a variant of *insertion sort*. The first k values are inserted into an array of k length, which is kept sorted through normal *insertion sort*. All subsequent values are compared to the values in the array in ascending order – if the current value is less than the smallest value (in the array) it is thrown away – if the current value is larger than the smallest value it is inserted at the proper position in the array and the smallest value in the array is thrown away.

This approach substantially limits the memory usage of finding the k largest values for each document vector. Whereas first calculating $\mathbf{A}'$ and then finding the largest values for each document vector would require $O(MN)$ memory, which would be unfeasible even for medium-sized document collections.

Returns: A table with k keywords (with corresponding weights) for each document.

6.1.3 Time and Memory Complexity

Normalisation

Finding and dividing by the lengths of the vectors are linear in the number of non-zero entries z in the sparse term-document matrix, so the computational complexity of this step is $O(z)$. The normalisation only requires enough memory to hold the term-document sparse matrix $\mathbf{A}$, since all operations are performed directly on this matrix. The memory complexity[3] is as a result also $O(z)$.

SVD

The computational complexity of the (reduced) SVD algorithm is usually related to the number of triplets calculated (LSA-dimension), since calculating the values of each singular triplet only depends on the preceding triplet [Ler99].

The SVD implementation used in our experiments[4] is a single-vector Lanczos method (`LANSO`). This is considered to be one of the fastest methods available today. Its computational complexity is $O(3Dz)$ [LFL98] and it has a memory complexity equal to the size of the resulting matrices[5] $O((N+M)D)$ [Bra02]. In addition matrix $\mathbf{A}$ also needs to be in memory during the calculations, so the real memory complexity is:

$$O(z) + O((N + M)D) = O(z + (N + M)D) \tag{6.4}$$

[3] Normalisation could probably be done one column at a time in memory while keeping the rest of matrix A on disk, thus resulting in an $O(M)$ memory complexity.

[4] See implementation details below.

[5] Σ is stored as an array of length D and it does (as a result) not contribute significantly to the memory complexity

Keyword extraction

The computational complexity of this step is dominated by the matrix multiplications ($\mathbf{U}'\mathbf{\Sigma}'\mathbf{V}'^{T}$) involved in constructing $\mathbf{A}'$ one column at a time from the singular triplets.

$\mathbf{U}'\mathbf{\Sigma}'$ is done first and has a computational complexity of $O(MD)$ (the number of values in $\mathbf{U}'$) due to the diagonal nature of $\mathbf{\Sigma}'$ – the values of each column of $\mathbf{U}'$ are multiplied by the column's corresponding singular value from $\mathbf{\Sigma}'$.

Multiplying the result of the above (which is still an $M \times D$ matrix) with $\mathbf{V}'^{T}$ has a computational complexity of $O(MND)$ (normal matrix multiplication).

The total computational complexity of this step is thus:

$$O(MND) + O(MD) = O(MND) \tag{6.5}$$

Since the number of terms M is expected to gradually stabilise in large document collections (with appropriate filtering for spelling mistakes and rare words etc.) and since the LSA-dimension D is a constant value, the algorithm approaches linear computational complexity in the number of documents.

The algorithm works on the result from the (reduced) SVD[6], so the memory complexity is $O((N + M)D)$.

6.1.4 Advantages

Since our keyword extraction scheme is custom designed to meet the needs of keyword based clustering, most of the advantages of the scheme have already been discussed in the above introductory section. Suffice to say that the scheme inherits many of the interesting properties of LSA and also ensures the required overlap of keywords that is needed to do subsequent clustering.

In addition, our scheme is not dependent on any linguistic information and is (like LSA itself) fully language independent. This is particularly important in connection with Mondosoft, since their existing products are designed to work "out of the box" with a wide range of languages.

Finally, it is important to remember that the extracted keywords might have other interesting applications than clustering. The scheme is by no means limited to clustering. The extracted keywords could for instance be used in addition to a text snippet when communicating the content of a page in a search result. One could also envision that the keywords might be used to classify documents into a given taxonomy, by linking particular keywords and combinations of keywords with each group in the taxonomy.

[6] The keywords and corresponding weights can easily be stored in a table on disk, so no memory is needed to store them.

6.1.5 Weaknesses

Our approach is based on a "bag-of-words" representation of documents and does as a result not utilise word order and the structure of the document. However, our implemented scheme for extracting bigram terms (discussed in the previous chapter) somewhat remedies this problem.

Another shortcoming of the above algorithm is that it returns a fixed number of keywords – it does not take the keywords' weight distribution into account. If the weights drastically drop off at a certain point, it might be beneficial to consider not saving any more keywords. This is especially an issue with clustering methods that do not take the weights of the keywords into consideration (e.g. lattice-based methods). As will be discussed in the section concerning implementation details below, we have chosen to implement a simple scheme that tries to remedy this shortcoming.

Our experiments have shown that the steps that calculate the (reduced) SVD and construct $\mathbf{A}'$ can be rather time consuming on large document collections (See section 9.2). However, the calculation of the columns (document vectors) of $\mathbf{A}'$ – which is the single most time consuming task – could easily be done in parallel and thus gain a significant speedup on multiprocessor machines.

In addition to the benefits of LSA, our approach also inherits some of the challenges of LSA discussed earlier (in section 3.2). It is for instance quite difficult to fine-tune the optimal LSA-dimension (number of singular triplets) and the approach is also quite sensitive to the preprocessing that is performed.

Appendix A documents some early tests of our algorithm on the CISI collection[7]. The tests illustrate how the LSA-dimension effects the extracted keywords. It can clearly be seen that the lower the LSA-dimension, the more general the keywords tend to get and the more keywords are inferred (even though they are not contained in the original document). The tests also show that for this collection, the 300 dimension value suggested by [LFL98] in some cases seems to be a bit too high – the keywords are very specific. However, we still believe that the 300 dimension value represents a fairly good compromise between being too specific and being too general for larger less homogeneous collections of real texts (not just abstracts). Later experiments on larger real text collections that have been subjected to the prepossessing (outlined in the previous chapter) seem to confirm this.

6.1.6 Implementation Details

We use Doug Rhode's publicly available software package SVDLIBC[8] to perform the (reduced) SVD. This package is fast and includes a simple-to-use interface, which among other things supports importing sparse and dense matrices from a variety of binary and text-based file formats. SVDLIBC is derived from Netlib's popular SVDPACKC[9] package, but offers a cleaned-up version of the code. Its SVD implementation is, as hinted above, based on the single-vector Lanczos method (`LANSO`). This method is among the fastest sparse matrix SVD methods available today. However, it has the drawback that the low order singular values may be relatively imprecise.

[7] A collection of 1,460 information science abstracts that can be downloaded in matrix form from `http://www.cs.utk.edu/~lsi/`.

[8] `http://tedlab.mit.edu/~dr/SVDLIBC`

[9] `http://www.netlib.org/svdpack/`

This is not really a concern to us, since we only need the D higher-order values and even in these we can tolerate some small degree of imprecision.

The above library uses double precision arithmetic for calculating the (reduced) SVD. However, we have created and tested a modified version of SVDLIBC that runs using single precision arithmetic. This was done to try to reduce memory consumption during SVD calculation on large matrices. Unfortunately, extensive testing showed that for certain very sparse input matrices, single precision was not enough to stabilise the required number of singular triplets (300).

The most time consuming step is, as already mentioned, the calculation of $\mathbf{U}'\mathbf{V}'^T$. We have made heavy use of profiling to ensure that our code performs reasonably well[10]. Understanding that matrix multiplication normally is a memory bandwidth bound problem, we have arranged $\mathbf{U}'$ and $\mathbf{V}'$ in memory in such a way that the processor's caches can be utilised in the best possible way. This is done by ensuring that the critical inner part of the multiplication loop works on values that lie in continuous segments of memory, thus utilising each cache-line optimally (see [Mar04]).

Keyword Cutoff Scheme

To remedy the problem of outputting the same fixed number of keywords for all documents regardless of the weight distribution of the documents' keywords, we have implemented a simple extension to the algorithm. The extension uses the weight distribution to determine how many keywords to output. The scheme works as follows:

Input:

- The minimum number of keywords k_{min} that the algorithm should return for a document. This minimum is included to ensure that all documents are assigned enough keywords to allow meaningful clustering.
- The maximum number of keywords k_{max} that the algorithm should return for a document.
- A cut-off ratio c (between 0 and 1). This value will be explained below.

Step 1: Perform the above outlined keyword extraction algorithm with the number of keywords set to:

$$k = \frac{k_{max}}{c} \tag{6.6}$$

[10] The profiling was done on our PowerPC-based Apple Powerbooks, since we did not have access to good profiling tools on the development x86 PCs provided by Mondosoft. However, we assume that most of our optimisations with regard to memory arrangement are more or less processor independent. Our code might still benefit from further x86-specific optimisations (e.g. loop unrolling and loop alignment) to increase the overall instruction throughput and avoid pipeline issue stalls. However, to properly evaluate the effects of such steps, a good profiler for the target architecture (x86) should be used.

Step 2: For each document; calculate the sum s of the k keywords' weights. Add the keywords with the highest weights to the table until k_{min} keywords have been added. Then repeat adding the remaining keyword with the highest weight to the table, until the accumulated sum of the added keywords' weights' is larger than $c * s$.

Returns: A variable number of keywords between k_{min} and k_{max} for each document, based on the weight distribution of the top k keywords.

This scheme is quite simple and better (more correct) solutions might exist, but the above method achieves the aim in an easy way that does not alter the overall keyword extraction algorithm's time and memory complexity significantly. Our testing on real world documents shows that the scheme works reasonably well in most cases with $c = 0.5$.

In appendix B the results of using our scheme (with $c = 0.5$ and $k_{min} = 0$) on different keyword weight distributions are illustrated to give the reader an impression off how it works in practise. k_{max} keywords are only returned if the weight distribution of the k keywords is more or less uniform.

6.2 Pure Truncation

In addition to the elaborate LSA-based keyword extraction scheme detailed above, we have also implemented a simple scheme based on truncation. This scheme is inspired by research presented in [SS97] that shows that pure truncation of terms (based on frequency/weight), leads to a dramatic reduction in clustering time without any significant reduction in cluster quality compared with full-profile clustering. An important reason for implementing this scheme, is that we need a baseline that can serve as a reference we can compare the LSA-based extractor against.

The truncation extractor simply assigns keywords by means of a truncation of the terms in a given document. This truncation is based on the frequencies/weights of the terms in such a way that the terms that occur the most or have the highest weight are assigned as keywords to a document. We also employ a cut-off scheme similar to the one used in the implemented LSA-based extractor, to ensure that the potential keywords' frequency/weight distribution is taken into account, when deciding how many keywords is assigned to a given document.

Chapter 7

Implemented Clustering Algorithms

In this chapter, we will give a thorough discussion of each of the five clustering algorithms that we have implemented. For each algorithm, we will first give a description of how and why the algorithm works, then we will discuss the algorithm's complexity, advantages and weaknesses. Finally, we will present relevant implementation details.

7.1 K-Means

7.1.1 Description

For a long time, K-Means has been considered the standard within clustering and still remains a very strong player in the field. The algorithm is very straightforward in that it simply partitions the documents in the term space into K clusters, hence the name. The algorithm thus produces a hard, non-hierarchical clustering. Typically, K-Means is used as an offline clustering algorithm for term spaces containing many documents. K-Means is based on the Vector Model, considering documents as vectors in multidimensional space.

In its most simple implementation, which is called Direct K-Means, K cluster prototypes are randomly selected from the documents in the term space. Hereafter, all documents are assigned to the cluster containing their closest centroid[1]. The cluster centroids are then calculated as the centroids of the documents in their respective clusters. The process repeats itself until a preset precision is achieved.

In order to achieve a hierarchical structure, we will demonstrate below how the algorithm is augmented to be bisecting.

7.1.2 Algorithm

Formally, the algorithm looks like this[2]:

[1] The centroid of a cluster C is the same as the vector-sum of all vectors i_l in the cluster divided by the number of vectors in the cluster - i.e. "the average document of the cluster": $\omega = \sum_{i_l \in C} \frac{i_l}{|C|}$.

[2] Adapted from [dSdSBJS$^+$04].

```
function Direct-K-Means($k, D$)
    (where $k$ is the number of desired clusters,
    and $D = \{i_1, ..., i_n\}$ is the set of documents to cluster)
    Initialise $k$ prototypes $(w_1, ..., w_k)$ such that $w_j = i_l, j \in \{1, ..., k\}, l \in \{1, ..., n\}$
    Each cluster $C_j$ is associated with prototype $w_j$
    Repeat
        for each input vector $i_l$, where $l \in \{1, ..., n\}$,
          do
            Assign $i_l$ to the cluster $C_{j*}$ with nearest prototype $w_{j*}$
        for each cluster $C_j$, where $j \in \{1, ..., k\}$
          do
            Update the prototype to be the centroid of all samples currently in $C_j$,
            so that $\omega_j = \sum_{i_l \in C_j} \frac{i_l}{|C_j|}$
          Compute the error function:
            $E = \sum_{j=1}^{k} \sum_{i_l \in C_j} |i_l - w_j|^2$
    Until $E$ does not change significantly or cluster membership no longer changes.
```

7.1.3 Time and Memory Complexity

Looking at the above function, we see the following complexities:

1. The initialisation is $O(k)$, where k is the number of desired clusters.

2. The main loop is repeated at most t times, where t is a set limit to prevent "live-locks" (where a document continually flips from one cluster to the other).

3. For each document, all distances must be computed to all prototypes/centroids, which gives us $O(kn)$, where n is the number of documents in the document space.

4. For each cluster, the centroids must be calculated, but since this is a hard clustering, we have only n documents to calculate this for and the complexity is thus $O(n)$.

5. Finally, the error function is computed, and again, this is done for all documents, and we arrive at $O(n)$.

The running-time complexity of the Direct K-Means is thus:

$$O(k) + t(O(kn) + O(n) + O(n)) = O(tkn) \tag{7.1}$$

K-Means has linear memory requirements, since we only need to store the documents, cluster membership of the documents plus the cluster centroids.

7.1.4 Advantages

The main advantage of K-Means and the reason for so many people advocating its use, is its very definite and good upper running-time and its simplicity. K-Means is near-linear, since t and k are nomally $\ll n$, which yields good performance in most applications.

7.1.5 Weaknesses

In spite of K-Means' popularity, quite a few issues exist that limit the advantages of using K-Means.

First of all, it is not at all clear how to determine the optimal number of clusters in a term-document space. If too few clusters are requested, unrelated data might end up in the same cluster, and conversely, if too many clusters are requested, the algorithm might split good clusters and place related documents in different clusters.

Secondly, K-Means in its default version is not hierarchical, which among other things makes it even more difficult to compensate for the problems caused by the cluster-number uncertainty. This also limits the advantages of K-Means for information retrieval purposes, since information will typically be expected to be organised in increasing complexity.

Furthermore, there is an uncertainty issue with regard to selection of the prototypes, the initial cluster centers. There is no way to know whether two centers are placed in well-separated clusters, or if they end up in the same cluster. This will of course result in clustering of poorer quality - partly because “tight” clusters might be split and partly because unrelated clusters might be combined. The turn-based centroid calculation will compensate for this to some extent, but it is still a “blind” algorithm which does not verify the quality of the created clusters or centers.

Finally, since K-Means is centroid-based in its cluster delegation, K-Means cannot be expected to discover clusters of non-convex shapes (i.e. not round or ellipsoidal) if clusters are not well-separated and at the same time comparatively dense or “tightly coupled”.

7.1.6 Implementation Details

The core algorithm is implemented as described above. However, we have designed a few modifications to make the algorithm more practical for our purpose.

Bisecting K-Means

A common approach to making partition-based algorithms hierarchical is to make the algorithm “bisecting” – instead of specifying the amount of clusters the core algorithm must generate, the core algorithm is required to generate two clusters, which are then put in a set of clusters. From this set, the largest cluster is chosen and two clusters are created from this using the core algorithm. This process is repeated, always choosing the largest cluster in the growing hierarchy of clusters, until the required amount of clusters is found. This also results in $O(tkn)$ complexity, but the constant values are somewhat larger. This is a price that we are willing to pay to get the hierarchy structure we need for the more-like-this functionality later on. We use a simple binary tree to store the hierarchical information, which has a worst-case storage complexity of $O(n^2)$ and a best-case storage requirement of $n\log n$, when we store the information about the contained documents in every node.

Bisection has a few disadvantages, the most noteworthy of these is an issue, when bisecting a low-dimensional space with dense clusters and relatively low inter-cluster distances. In this case, actual clusters might be split into each part of a bisection, an example of this is shown in appendix C (the colours represent distinct clusters). This should not have too great an impact within our document space, since the document space is so sparse that unrelated clusters most likely have ortholognal axes compared to the clusters being split.

Data Representation

As outlined earlier, the documents are represented by keywords, where each has a weight. Since the keyword extraction removes most of the terms from each document, a document vector is largely "empty" or sparse, composed almost entirely of zero-weight terms. Therefore, representing the vectors as sparse vectors, where only keywords and with non-zero weights are specified, will save a lot of memory.

In the initial stage of the clustering algorithm, we have chosen to order the keywords of each individual document vector in ascending order[3]. Although this requires running quicksort [CLRS01, pp. 145] on every single vector, this will save billions of search operations, when the distances are calculated between vectors later in the process. Since the data representation for documents cannot be sorted using the ANSI C `qsort` implementation, we have made a custom implementation of quicksort.

The clustering is (besides from being stored in a binary tree) mainly performed "in-place" in an array of document references in order to quickly access the data and avoiding the penalty from searching the binary tree for the largest leaf. Information about the cluster positions in the array is stored separately.

Distance Measure

As seen in section 2.2.4, measuring distance in high-dimensional spaces by for instance the *Euclidean distance measure* or the *Manhattan distance measure*, will often result in distances of approximately the same length that will not be well-suited for clustering purposes. To slightly amend this problem which is related to the curse of dimensionality, we have chosen to use the *Cosine distance measure*, which measures the angular separation of documents as recommended in [BG04][4], since only the directions of the document vectors and not their lengths are important. Equation 7.2 demonstrates the simplicity of the cosine distance measure:

[3] In our system, every keyword/term is assigned a unique ID and we sort the vectors according to this.

[4] Furthermore, by using the cosine distance measure, we save a lot of calculations since the dot product of two vectors in our very sparse keyword space will most often be zero.

$$cos(A, B) = \frac{A \cdot B}{|A||B|} \tag{7.2}$$

In order to be able to exchange the cosine distance measure with other measures, such as the Euclidean, we calculate *acos* of the result to get the distance in radians. This will give us zero, when the document vectors are parallel and the documents thus identical or highly similar and $\frac{\pi}{2}$ when the documents have nothing in common[5].

Since the vectors are sorted ascendingly according to the keyword id, we can implement the cosine distance calculations between document A and document B in a very efficient manner:

```
function cos(A, B)
        $i = 0, j = 0, cos = 0.0$
        Repeat
                if $A.keywordID[i] == B.keywordID[j]$ do
                   $cos = cos + A.weight[i] * B.weight[j]$
                   $i = i + 1, j = j + 1$
                else if $A.keywordID[i] < B.keywordID[j]$ do
                   $i = i + 1$
                else do (i.e. $A.keywordID[i] > B.keywordID[j]$)
                   $j = j + 1$
        Until $i == A.length$ or $j == B.length$
        Return $acos(cos)$
```

An algorithm following the same pattern can be made for the Euclidean distance measure as well as others, if so desired.

Database Structures

Since we have a complete hierarchy of all performed bisections, we have each document not only in its base (or, leaf) cluster, but also in the previous, larger clusters. Since we want to be able to use various amounts of results, and since we cannot guarantee a specific size of the smallest clusters, we have decided to store all the clusters from the hierarchy in the database.

This means that if the document A is in the base cluster 1, which was earlier split from cluster 2, we store references to both cluster 1 and cluster 2 in A's database entry and, conversely, we store both cluster 1 and cluster 2 in the database (and both have references to document A). The structure of the database entries are shown in figure 7.1:

Future Improvements

Since the selection of cluster centers still isn't quite optimal (i.e. randomised), RPCL [KL99], a simpler clustering algorithm, which is especially practical and feasible when only requiring few

[5] Actually, some of the term weights for a given document could in rare cases be made negative from running SVD in the keyword extraction phase, which could result in a cosine distance up to π. Our algorithm will also function correctly in this interval.

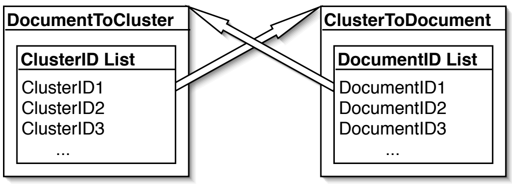


Figure 7.1: *The structures used to save the clusters in the database.*

clusters, could be used to find initial centers of higher quality. These initial centers are then fed into the bisecting K-Means algorithm. This will either result in shorter running time (fewer iterations of the core loop), fewer wrongful splits or both.

In order to make the algorithm practical for very large data sets (millions+ of documents), the binary tree should be reconstructed so that it only stores document references in its leaves. This will make its storage requirements linear, and the storage requirements of running the K-Means algorithm will then be $O(kn)$ in the worst case.

7.2 CURE

7.2.1 Description

CURE, Clustering Using REpresentatives, is a somewhat more complicated approach to a clustering algorithm than K-Means. It is hierarchical and agglomerative (or "bottom-up") by design, which means that it basically works by joining smaller clusters to form larger clusters. The clustering is "hard" in that a document is only present in a single cluster. CURE is often commended for its ability to detect non-convex shapes as opposed to the more popular K-Means. As we shall see, CURE is a very complex algorithm and we thus consider it an offline algorithm.

Upon initialisation, every document in the set is considered to be a cluster. The algorithm determines the closest pair of clusters and joins these together forming a new cluster and repeats this until the number of clusters is "low enough". In order to save memory, computations and to prevent the algorithm from being influenced by noise in the clustering process, at most c *representatives* are used to represent a cluster. The default value for c is typically 10 as defined in [GRS98]. These representatives are chosen as far from the center as possible as well as as far from each other as possible. To prevent outliers from entering the cluster too early, the representatives are then shrunk by the factor α towards the center. α is typically set between 0.2 and 0.7 as also recommended in [GRS98].

The distances between clusters A and B is defined to be the minimum distance between their representatives[6]:

$$dist(A,B) = min\{dist(p,q) : p \in u.rep, q \in v.rep\} \tag{7.3}$$

Figure 7.2 illustrates how CURE performs its clustering.

[6] This distance measure is also referred to as *single-link* or *minimum-link* [FBY92].

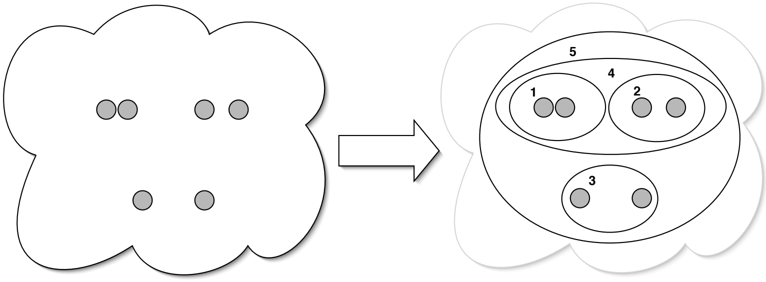


Figure 7.2: *The order in which CURE merges its cluster hierarchy.*

7.2.2 Algorithm

Formally, the overall algorithm for CURE looks like this:

```
function cure($k, D$)
        (where $k$ is the number of desired clusters,
        and $D$ is the set of documents to cluster)
        $T$ = build_kd_tree($D$)
        ($T$ contains the representatives for each cluster)
        $Q$ = build_heap($D$)
        ($Q$ contains the actual clusters)
        while size($Q$) > $k$ do
              $u$ = extract_minimum($Q$)
              $v = u.closest$
              delete($Q, v$)
              $w$ = merge($u, v$)
              delete_representatives($T, u$); delete_representatives($T, v$)
              insert_representatives($T, w$)
              $w.closest = x$ ($x$ is an arbitrary cluster in $Q$)
              for each $x \in Q$ do
                  if dist($w, x$) < dist($w, w.closest$) do
                     $w.closest = x$
                  if $x.closest$ is either $u$ or $v$ do
                     if dist($x, x.closest$) < dist($x, w$) do
                        $x.closest$ = closest_cluster($T, x$, dist($x, w$))
                     else do
                        $x.closest = w$
                     relocate($Q, x$)
                  else do
                     if dist($x, x.closest$) > dist($x, w$) do
                        $x.closest = w$
                        relocate($Q, x$)
              insert($Q, w$)
        Return($Q$)
```

CURE uses two data structures, a heap (see below) Q and a KD-tree[7] T that are first initialised with the documents from D as clusters. Q is ordered according to the distance to the closest cluster.

After forming the heap and the KD-tree, the algorithm enters its main loop. In the loop, the closest pair of clusters, u and v, are extracted from Q and merged to form a new cluster, w. T is then updated to reflect this change. Since any other cluster in Q may still have a reference to either of the removed clusters, every cluster in Q must be checked and possibly updated reflecting the change.

The KD-tree is used to find the closest cluster in the case where w is further from a cluster than u and v were. Whenever $x.closest$ is changed, Q is reordered to maintain the heap property of Q.

Finally, w is inserted into Q and the loop continues until k is reached.

CURE's merge operation is not the same as simply making the union of two clusters, since we use representative points for the clusters. The merge-operation looks like this:

```
function merge($u, v$)
    $w = u \cup v$
    $w.mean = \frac{|u|u.mean+|v|v.mean}{|u|+|v|}$
    $tmpSet = \emptyset$
    for $i = 1$ to $c$ do
        $maxDist = 0$
        for each document $p \in$ cluster $w$ do
            if $i == 1$ do
                $minDist =$ dist$(p, w.mean)$
            else do
                $minDist = min\{$dist$(p, q) : q \in tmpSet\}$
            if $minDist \geq maxDist$ do
                $maxDist = minDist$
                $maxPoint = p$
        $tmpSet = tmpSet \cup \{maxPoint\}$
    for each point $p \in tmpSet$ do
        $w.rep = w.rep \cup \{p + \alpha(w.mean - p)\}$
    Return $w$
```

First, the document sets are joined. Hereafter, the mean is calculated for the new cluster. This corresponds to calculating the *centroid* in K-Means, but the mean plays a smaller role in this clustering scheme.

In the first iteration of the outer for-loop, the document with the longest distance from the mean is selected as a representative. In subsequent iterations, the chosen representatives must have the longest distance from the representatives already in *tmpSet*.

[7] A multidimensional search tree for points in a k-dimensional space. Levels of the tree are split along successive dimensions at the points. Also see `http://en.wikipedia.org/wiki/Kd-tree`.

Finally, the representatives are “shrunk”, as outlined in the description, by the factor α towards the mean and the new cluster is returned.

7.2.3 Time and Memory Complexity

Guha, Rastogi and Shim, the authors of the CURE algorithm, specify in [GRS98] that CURE’s core algorithm has a worst-case complexity of $O(n^2 \text{log} n)$ for high-dimensional spaces. To remedy this, a sampling scheme and a partitioning scheme are suggested to decrease the complexity of the problem.

First, a “good” random sample is taken from the total document set. The sample size can be calculated using “Chernoff Bounds” [GRS98]. Hereafter, the sample is partitioned into large sets using some standard partitioning algorithm for instance a simple K-Means partitioning. The sets are then individually clustered using CURE until the final number of clusters in each partition reaches $\frac{n}{pq}$, where n is the number of documents in the sample, p is the number of partitions and q is a chosen constant larger than 1. Then the sets are joined and the clustering is continued until the desired number of clusters is created. Finally, the remaining documents (those that were not taken into the sample) are put into corresponding clusters based on their proximity to the clusters’ representatives.

The effects of reducing the amount of documents to be clustered on the running-time are as given above. Partitioning will according to [GRS98] provide a further improvement factor of $\frac{q-1}{pq} + \frac{1}{q^2}$[8].

The heap and the KD-tree both have linear complexity when storing the documents or clusters. Keeping track of the hierarchy requires an additional entry for every cluster merge in some data structure, which will also have linear memory requirements.

7.2.4 Advantages

CURE offers some very interesting features such as the ability to actually cluster non-convex shapes due to the way that the cluster representatives are chosen. Furthermore, the way the clusters are generated seems like a more intelligent approach, since no degree of randomisation is included in the actual clustering process[9].

In addition, the shrinking scheme should help avoid the chaining situation[10], which is an unfortunate side-effect of using the single-link distance measure. Compared to the all-points approach of *Minimum Spanning Tree* [ZHTY03], using representatives for clusters helps reduce the overall running time of the algorithm.

As before mentioned, the clustering is hierarchical by design, so the structures generated by CURE should be easily traversable.

[8] Unfortunately, when calculating the sample size and considering the fact that we need very small clusters for our more-like-this system, we have concluded that it is not sensible to perform sampling or partitioning.

[9] Of course there is the very unlikely situation where a cluster has two or more closest clusters plus the situation where more representatives of equal maximal distance exist.

[10] Described in section 3.1.2.

7.2.5 Weaknesses

The worst-case running-time of CURE does not seem promising. Even with random sampling, for a large document collection (100,000+) the Chernoff Bounds [GRS98] work out sample sizes so large (see appendix D) that the running time is problematic.

Another problem is that a certain minimum size for clusters needs to be defined when using sampling. This will inevitably result in more of the clusters being too large and covering more than one concept when the sample clustering completes, than if we had not applied sampling.

7.2.6 Implementation Details

Since our test data sets are comparatively small [GRS98] and due to the problems with minimum cluster size mentioned above, we have decided not to implement sampling and partitioning. The distance measure applied is the same as that of K-Means, namely the cosine distance measure.

In CURE's internal representation, we have implemented a heap (see below) that CURE uses to extract the nodes to merge from. Each node in the heap is a cluster.

After run-time profiling the algorithm without first implementing the KD-tree, we determined that the performance gain from implementing a KD-tree would only amount to 2-3% of the total running time of CURE on the data sets that we used. So we decided to skip this data structure and thus save the memory otherwise required for storing the KD-tree .

Heap

A heap node contains information about its distance to origo (to speed up distance calculations), its representatives, a reference to its closest node, references to all documents contained in the cluster and finally a binary tree representing the order in which documents were added to the cluster.

For each iteration of CURE's main loop, the cluster with the closest other cluster is extracted. This operation is simple enough, since we have sorted the heap to have the cluster in the top. However, later steps require us to relocate nodes in the heap. There is to our knowledge no default method for relocating nodes in a heap, so we have designed an algorithm ourselves that in linear time can remove and reinsert any node from a heap and preserve the heap property[11].

The principle of the algorithm is fairly straightforward (we will use the term tree in this explanation, to help the reader better understand the structure of the heap):

1. Locate and remove the desired node.
2. Take the right-most leaf from the lowest level of the tree and place it in the vacated spot in the tree. Now, we can no longer be sure that the heap property is preserved for the tree

[11] The heap that we work on is a min-heap (lowest in the top). The min-heap property states that each node in a tree has a key which is greater than or equal to the key of its parent. Also see `http://www.nist.gov/dads/HTML/minheapprop.html`. Furthermore, we always want to fill the tree left-to-right, top-to-bottom and preserve that structure. (i.e. no holes).

below the new node, nor for the branch extending from the root to the node. Therefore,

3. Move the node up in the tree by swapping it with its parent until the node is greater than its parent. If this could be done, we know that any nodes deeper in the tree will invariably be greater than the node (since they were greater than their parents, who were greater than the node).
4. However, if the node is not smaller than its initial parent, then either of the node's children may be smaller than the node, thus, the node is swapped with the smallest of its children until the node is smaller than both of its children. Now the entire tree/heap has the heap property.

Thus heap deletion at any place in the heap can be done with linear complexity. The complexity is linear because the search for locating the node is executed in linear time. After the deletion is done, reinsertion can quickly be performed to put the node in its proper place.

Database Structures

Since CURE also generates a hierarchical structure, we have decided to use the same structures as K-Means (see section 7.1.6 - *Database Structures*) for storing the clusters in the database.

Future Improvements

Since we decided to skip sampling and partitioning, we haven't been able to test the quality of clustering using these techniques. However, according to the author of the original algorithm, CURE still provides clusters of decent quality. It would therefore be interesting to try to implement both of these to see if the speed-up gained is worth the quality lost. Furthermore, outlier elimination should also be considered, since this could also improve cluster quality.

Since we get a performance penalty from reordering the heap, future improvements should also focus on either choosing a different data structure or on changing the data representation to make it possible simply to perform the reordering without first removing the node.

Even though CURE is agglomerative, it still tends to favor the hyper-spherical clusters due to the centroid calculated when choosing the cluster representatives and the shrinking towards this. In order to avoid this, a different shrinking scheme has been suggested in [QSW02]. Implementing this would possibly yield clusters of higher precision.

7.3 PDDP

7.3.1 Description

Like our bisecting implementation of K-Means[12], PDDP (Principal Direction Divisive Partitioning [Bol98]) is a hierarchical algorithm. But where K-Means was made hierarchical by bisection,

[12] See section 7.1.

PDDP is hierarchical by design. The way the partitioning takes places differs by a lot, however.

PDDP uses a 1-dimensional singular value decomposition[13] (SVD, see section 3.2.2) to determine the principal direction of a document set, and thereafter splits the set into two subsets according to their projection on the principal direction. The sets are placed in a simple binary tree and a "scatter value" is calculated for each node in the tree as they are created in order to determine which leaf the algorithm will split next. The algorithm terminates when enough clusters have been formed in the tree.

Thus, the algorithm is hierarchical by design. Furthermore, as can be seen from above, this algorithm only produces hard clusters like those of K-Means and CURE. Finally, whereas the algorithm is intended to be used offline, it is fast enough for online clustering of small document sets.

7.3.2 Algorithm

```
function PDDP($M, C_{max}$),
    (where $M$ is a matrix of document vectors and $C_{max}$ the number of desired clusters.)
    $root$ = initialise_pddp_tree($M$)
    while $C_{max} > 1$ do
        $split$ = get_most_scattered($root$)
        create_empty_left_child($split$)
        create_empty_right_child($split$)
        $\omega = \frac{1}{size(split)} \sum_{i=1}^{size(split)} split.documents[i]$ (the centroid of the node)
        $U\Sigma V^T$ = perform_1Dimensional_SVD($split.documents$)
        $\mu$ = extract_first_column($U$)
        for $i = 1$ to $split.size$
            $\sigma\nu = \mu^T(split.documents[i] - \omega)$ (where $\sigma$ is a positive constant)
            if $\nu \leq 0$
                add($split.left, split.documents[i]$)
            else
                add($split.right, split.documents[i]$)
            $split.left.scattering$ = calculate_scattering($split.left$)
            $split.right.scattering$ = calculate_scattering($split.right$)
    Return $root$
```

Note that C_{max} actually means the number of desired leaves of the tree, but since we are interested in all the levels of the tree, the actual number of clusters, we produce will be $2C_{max}$.

As we can see above, the core concept of the PDDP algorithm is determining ν, which is the projection of the document on the principal direction. Since the vectors are "normalised" using the centroid of the term space, we can easily tell in which of the new clusters the document should be added by the projection's relative position to origo.

The scattering measure of a cluster partition M_p (with centroid ω) is equal to the Frobenius

[13] In the original PDDP the leading eigenvector of the covariance matrix is used. However due to the computational complexity of this, we use a 1-dimensional SVD to estimate the eigenvector.

norm of the corresponding matrix $A = M_p - \omega e^T$, where $e = (1, 1, ..., 1)^T$ and the length of e is the same as the amount of documents in M_p. The square of the Frobenius norm is defined as such:

$$\|A\|_F^2 = \sum_{i,j} |a_{i,j}|^2 \tag{7.4}$$

Thus, calculating the scattering value requires calculating the corresponding matrix, adding all elements squared and taking the square root of the result.

The scatter value of the cluster reflects the distance of each document in the cluster to the centroid of the cluster and is thus an efficient measure for determining the next cluster to split. Note that the scatter value of the cluster is the only component of the algorithm which is based on a distance measure [Bol98, GLY03] and different scattering measures could just as easily be applied.

7.3.3 Time and Memory Complexity

Analysing the above function, we see the following complexities:

1. The main loop of the algorithm is executed C_{max} times.
2. Searching the tree for the most scattered node requires at most n lookups where n is the number of documents in the set.
3. Calculating ω requires n vector additions, where each vector has at most z non-zero elements.
4. The 1-dimensional SVD can be performed in $O(nz)$ (see 3.2.2).
5. Finally the scattering of the two new clusters can also be calculated in $O(nz)$ as seen above.

Thus, the worst-case running time of the PDDP algorithm is:

$$C_{max}(O(n) + O(nz) + O(nz) + O(nz)) = O(C_{max}nz) \tag{7.5}$$

The storage requirements of the PDDP tree are the same as those of the binary tree used by bisecting K-Means, namely $O(n^2)$ in the worst-case and $n\log n$ in the best case.

7.3.4 Advantages

As can be seen above, PDDP has a better worst-case running time than K-Means, since z, the number of non-zero elements in a given document vector, is typically a lot smaller than t, the number of times the K-Means loop must execute before the precision is good enough.

Furthermore, the scattering value enables us to more efficiently determine when the algorithm should stop – it is pointless to split clusters that consist of almost-identical documents. Using the scattering value, we can avoid splitting possibly large, but dense clusters, and at the same time make sure to split small, but diverse clusters.

Since the PDDP algorithm already works on a tree, we get the benefits of the tree structure without adding additional computational requirements.

7.3.5 Weaknesses

Calculating the scattering values makes the clustering algorithm very precision dependent, thus we might expect some clusters that are not completely dense, but dense compared to the surrounding term space to end in an unpartitioned cluster if the precision is not high enough to detect a scattering.

Since we simply split the term space in the middle of the principal direction, it is unclear whether the place we split will be in the middle of an otherwise dense cluster, or if it will be between clusters as we desire.

7.3.6 Implementation Details

The PDDP algorithm is implemented as outlined above. Like both CURE and K-Means, PDDP operates on sorted sparse keyword vectors with the exception that PDDP uses dense vectors when working with the vectors ω, ν and μ.

SVD

In order to calculate the principal direction of the term space, we use the same SVD library that we use for extracting keywords in section 6.1 (SVDLIBC). Since the algorithm it utilises is iterative, we can calculate the principal direction in very few steps.

The problem with using SVDLIBC is that there is no default method of sequentially adding vectors to a sparse matrix, and if we were to use dense matrices, the storage requirements could quickly go above the 32-bit limit of modern computers or at the very least require immense paging of data to and from the hard disk. In order to solve this, we have created a data structure that "wraps" around the default sparse matrix data structure with additional information about memory allocation. This enables us to efficiently add sparse vectors to the matrix in such a way that we don't use reallocation operations all the time and nor do we waste extra space by using dense matrices.

Database Structures

PDDP generates a hierarchical structure per default, and we have decided to use the same structures as K-Means (see section 7.1.6) and CURE for storing the clusters in the database.

Future Improvements

As is the case for K-Means – in order to make storage requirements of PDDP linear in the number of documents in the term space, we could change the tree structure such that document references are only kept in the leaf nodes of the tree. This will not have a performance penalty when clustering, but may yield some difficulties when storing the clusters in a database.

In order to increase the quality of clusters, PDDP should be reimplemented using 64-bit precision, when computers supporting this become available. This will allow for splitting clusters that are more dense than we are currently able to split.

Finally, the author of PDDP [Bol98] mentions that the document space may be split using other criteria than simply whether $\nu \leq 0$, for instance origo could be translated to a least-dense part of the term space in order to make splits of higher quality.

7.4 GALOIS

7.4.1 Description

GALOIS is a conceptual clustering method that generates soft clusters arranged in a lattice structure. The algorithm automatically generates what is known as a Galois lattice from a given set of objects – documents in our case – each with a set of attributes – keywords in our case. It was invented by Carpineto and Romano and is detailed in [CR96] and [CR95]. However, before we go into details with GALOIS, we will briefly provide an overview of the main ideas and theory behind conceptual clustering methods.

Conceptual Clustering

Conceptual clustering methods are closely related to *Formal Concept Analysis* (FCA), an emerging area that has many promising applications within data analysis, information retrieval and knowledge discovery [KB02]. The central notions in FCA are *formal context* and *formal concept.* A *formal context* $\mathcal{K}$ is modelled as a triple:

$$\mathcal{K} := (G, M, I) \tag{7.6}$$

Where G is a set of objects, M is a set of attributes and I constitutes a binary relation between G and M ($I \subseteq G \times M$). If an object g has attribute m then $(g, m) \in I$.

In the *formal context* (G, M, I) a *formal concept* is defined as a pair (A, B) with $A \subseteq G$ and $B \subseteq M$. A is called the *extent* of the formal concept and B the *intent*. The *extent* (A) of the formal concept describes the subset of objects that belong to the concept, while the *intent* (B) describes0 the common attributes that these objects share:

$$A := \{g \in G \mid \forall m \in B : (g, m) \in I\} \tag{7.7}$$

$$B := \{m \in M \mid \forall g \in A : (g, m) \in I\} \tag{7.8}$$

The *extent* and the *intent* are thus dual notions each fully defining the formal concept. The formal concepts can be ordered by applying the standard set inclusion to the *intents* (or dually to the *extents*) naturally forming a partial order that defines the *subconcept-superconcept relation*:

$$(A_1, B_1) \leq (A_2, B_2) :\Longleftrightarrow A_1 \subseteq A_2 \quad (\Longleftrightarrow B_2 \subseteq B_1) \tag{7.9}$$

A concept C_1 is hence a *subconcept* of another concept C_2 ($C_1 \leq C_2$) if the objects (extent) of C_1 are a subset of the objects (extent) of C_2 and the attributes (intent) that define C_2 are a subset of the attributes (intent) that define C_1. C_2 is thus more general than C_1, encompassing all the objects (the extent) of C_1, while C_1 is more specific than C_2, encompassing all the attributes that define C_2 (the intent of C_2).

The set of all formal concepts in a formal context $\mathcal{K}$ with the partial order $\leq$ (as defined above) forms a conceptual hierarchy. Mathematically, this hierarchy turns out to be a *complete lattice*. This resulting lattice is often called a *Galois lattice*[14] [HSS03]. A Galois lattice thus represents a conceptual hierarchy of the objects in a given formal context, based on the object's attributes within that context. See figure 7.3 for an example of a Galois lattice. Orderings of concepts, like the one represented by a Galois lattice, play a fundamental role in the process of organising data, information and knowledge [NO02]. They are often used to represent latent conceptual hierarchies that are implicitly present in the underlying data [KB02].

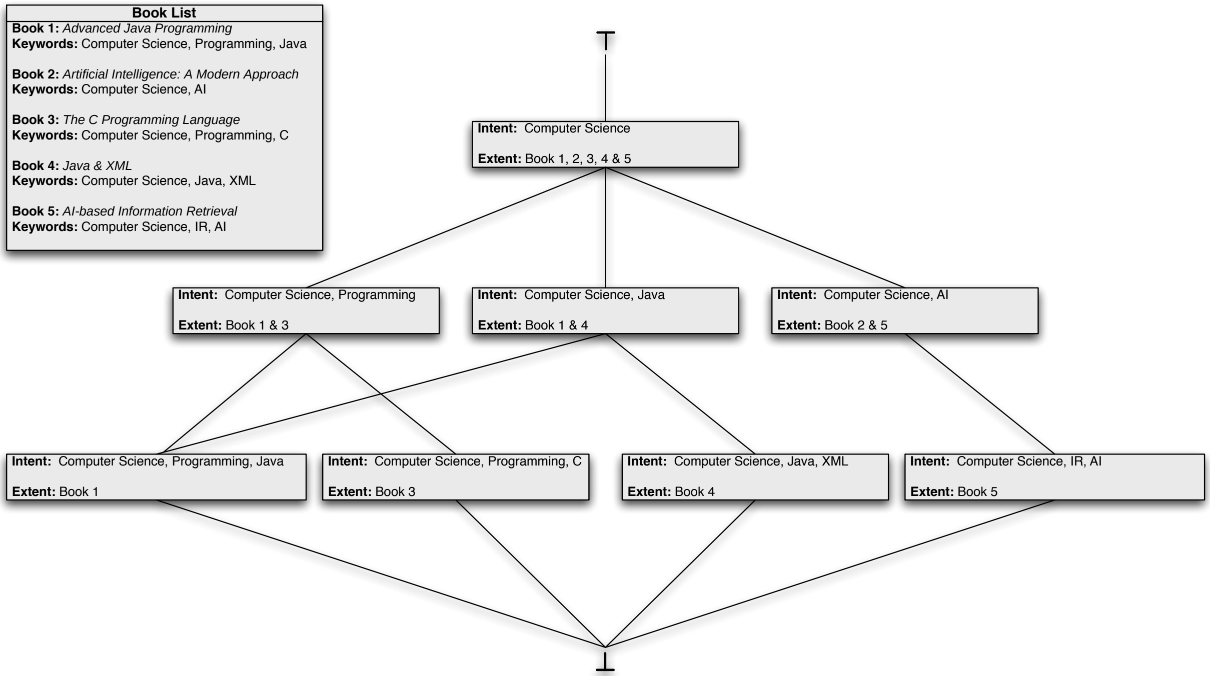

Book List
Book 1: Advanced Java Programming
Keywords: Computer Science, Programming, Java
Book 2: Artificial Intelligence: A Modern Approach
Keywords: Computer Science, AI
Book 3: The C Programming Language
Keywords: Computer Science, Programming, C
Book 4: Java & XML
Keywords: Computer Science, Java, XML
Book 5: AI-based Information Retrieval
Keywords: Computer Science, IR, AI
⊤
Intent: Computer Science
Extent: Book 1, 2, 3, 4 & 5
Intent: Computer Science, Programming
Extent: Book 1 & 3
Intent: Computer Science, Java
Extent: Book 1 & 4
Intent: Computer Science, AI
Extent: Book 2 & 5
Intent: Computer Science, Programming, Java
Extent: Book 1
Intent: Computer Science, Programming, C
Extent: Book 3
Intent: Computer Science, Java, XML
Extent: Book 4
Intent: Computer Science, IR, AI
Extent: Book 5
⊥

Figure 7.3: *An example Galois lattice built from a small collection of books (with assigned keywords)*

[14] The lattice is sometimes also simply referred to as the *concept lattice*.

Compared with other forms of hierarchical orderings (e.g. tree-like structures), lattices have some distinct advantages. A lattice is a special kind of partial ordering that have some nice algebraic properties. The connecting result (see [NO02]) states that any lattice can be conceived of as either a partial order or as an algebra, or as both. These two ways of defining a lattice can be used in an entirely interchangeable way, depending on which of them appears to be more convenient for a particular purpose. When used in algebraic mode, Galois lattices allow "calculations" with concepts in a strictly mathematical way [NO02]. One can for instance (in an entirely algebraic way) find the *least upper bound* of 2 concepts[15] as the intersection of the concepts' intents or the *greatest lower bound* that 2 concepts share[16] as the union of the concepts' intents.

Another advantage of lattices, compared with tree-like hierarchies, is that lattices permit *multiple-inheritance* allowing a concept to be a subconcept of several other concepts. This, for instance, enables us to model "lattice" as a subconcept of both "partial order" and "algebra". For clustering purposes, *multiple-inheritance* allows the cluster of documents related to "software engineering" to be a sub-cluster of both the documents related to "software" and the documents related to "engineering".

GALOIS

As touched upon in the beginning of this section, GALOIS generates concepts (clusters of objects) by automatically generating a Galois lattice (as defined above) from a set of objects and their corresponding attributes. In the context of document clustering, the objects are documents and the attributes are keywords (or index terms) assigned to these documents. The clusters resulting from GALOIS are thus ordered in a lattice and characterised by both an extensional description (i.e. the documents in the cluster) as well as an intentional (conceptual) description (i.e the keywords that these documents share). A cluster's extent consists of precisely those documents that contain the cluster's intent as a subset of their assigned keywords. Each cluster is hence uniquely defined by either a set of documents (extent) or a set of keywords (intent). Additionally, a cluster's "parents" (if any) are always defined by a subset of the set of keywords that defines the given cluster and always contain super-sets of the documents in the given cluster.

The distance measure applied for document clustering using GALOIS is as a result simply the inverse of the number of shared keywords (found using intersection). The more keywords two documents share, the closer they are conceptually, since they will co-occur in the extents of more specific concepts/clusters.

Besides GALOIS, there exist several other approaches to constructing a Galois lattice from a set of objects and their corresponding attributes. [KB02] provides a good overview of some these other approaches. However, we have chosen to implement GALOIS since it is relatively fast compared with other common approaches and at the same time incremental - meaning that the Galois lattice can be constructed one document at a time and later expanded as new documents are found/created [CR96].

GALOIS defines the incremental step of adding a new document to an existing (possibly empty) Galois lattice. The first thing we notice is that existing concepts are never removed from the

[15] Called *supremum* or *join* in lattice terminology.

[16] Called *infimum* or *meet* in lattice terminology.

lattice – once formed, concepts (nodes in the lattice) are permanent and their intents remain immutable. However, new objects (documents) might be added to existing concepts thus expanding their extent. The addition of a new object might also give rise to the generation of new concepts (lattice nodes). This happens when the intersection of the new object's intent with the intent of any existing set of objects, with which the new object shares a common description (intent), is not already present as a concept in the lattice. When new concepts are generated in this way, they have to be consistently linked into the existing lattice structure at the proper place. This process might both generate new edges and remove edges between concepts. For each new object introduced into the lattice, the 3 main tasks of the algorithm is thus to[17]:

1. Add the object to any existing concepts (nodes) where it belongs (i.e. the concept's intent (keywords) are a subset of the document's).
2. Generate all new concepts in the lattice that the new object gives rise to, avoiding creating concepts that are already present in the lattice
3. Update the linking structure (the edges) of the lattice to reflect the added concepts.

To obtain the new concepts that need to be created, it is sufficient to consider the intersections of the new object's intent with the intent of each concept currently in the lattice [CR96]. We use *the connecting result*[18] discussed earlier:

> The intent of the *least upper bound* of two concepts is the intersection of the intents of the two concepts.

This upper bound represent a generalisation that encompasses both concepts. In connection with adding a new object (document), the intersection hence represents a concept where both the new object and the objects of the existing concept belong. If this concept doesn't already exist it needs to be added.

The new object and the existing concepts are, however, not compared independently. To quote [CR96]:

> "Roughly, GALOIS compares the parents of each node [concept] to the intersection (of intents) of the node and the new object. If there is a parent with an intent that equals (=) the intersection, then the new concept is not created because it is already present in the lattice. If there is a parent with an intent that is a proper superset ($\subset$) of the intersection, then the new concept is not created since it will be generated when we intersect the new object with that parent of the node."

However, if all parents of a given concept (node) have intents that are proper subsets of the intersection (of that concept and the new object) or if all parents are incomparable to the intersection, a new concept is added into the lattice. We thus only have to consider the intent of existing concepts and the intent of the new object (document) when we determine where to

[17] Adapted from [CR96]

[18] See [NO02]

add the object and which new concepts (nodes) to create (if any). This makes the algorithm quite simple since it mostly operates on intents.

Galois uses the `link` procedure outlined below to link new concepts into the lattice. `link` operates on the lattice as it is at the time the new concept is added, also taking concepts that have been created as part of inserting the current object into account. `link` works by determining two boundary sets for the new concept (node):

1. The upper boundary set G that contains the most specific concepts that are more general than the new concept.
2. The lower boundary set S that contains the most general concepts that are more specific than the new concept

`link` then links the new concept (node) to all elements in S and G (As child of concepts in G and as parent of concepts in S). `link` removes all edges between S and G. A formal proof of the correctness of the GALOIS algorithm can be studied in [CR96].

7.4.2 Algorithm

Formally, the GALOIS algorithm for adding a new object (document) to the lattice looks like this[19]:

```
function update-lattice(new object object, lattice lattice)
        If not already present, add the concept corresponding to object to lattice
        for each concept $C_i$ in lattice,
            do
               Find intersection as $C_i.intent \bigcap Object.keywords$
               unless
                      $intersection = \{\text{empty set}\}$, or
                      $intersection = C_i$, or
                      There is $\text{parent}(C_i) = intersection$, or
                      There is $\text{parent}(C_i) \supset intersection$
               then
                     Create a new lattice node newnode
                     $newnode.intent = intersection$
                     $newnode.extent = object \cup C_i.extent$
                     $lattice = \textbf{link}(newnode, lattice)$
               if
                  $intersection = C_i$
               then
                     $C_i.extent = object \cup C_i.extent$
        return the updated lattice
```

[19] Adapted from [CR96]

```
function link(new node node, lattice lattice)
        Find upper boundary set $G$ of node in lattice
        Find lower boundary set $U$ of node in lattice
        Add edges between node and each element in $G$
        Add edges between node and each element in $U$
        Eliminate edges between $U$ and $G$
        return the updated lattice
```

7.4.3 Time and Memory Complexity

The number of concepts C in the Galois lattice is central to both the computational complexity and the memory complexity of adding a single document using the above algorithm. There is a theoretical upper bound on the number of concepts present in a given lattice, which depends on the number of documents and the size of the object description space (i.e. number of keywords per document, total number of unique keywords assigned to the documents etc.). See the lengthy discussion in [CR96]. However, this theoretical bound is extremely high and does as a result have very little practical value. Empirical values of C are thus needed to estimate the actual space requirement and running time of the algorithm.

[CR95] found empirical and theoretical evidence that the size of the lattice (number of concepts C) on average grows almost linearly with the number of documents N. However, several experiments carried out in [CR96], showed that the growth in the number of concepts varies from linear to quadratic in the number of documents, but in some cases drops of as a result of the domain starting to become saturated. Saturation in this connection implies that most of the naturally occurring concepts within the domain are already present in the lattice. This saturation effect thus depends on the conceptual span of the document collection.

According to our own tests, the LSA-dimension used to extract keywords very much influences C as well (see appendix J.7.6). Very low LSA-dimension resulted in fast saturation and thus small values of C due to the generalising effect of LSA, this meant that a relatively small set of keywords were used to describe all documents. In the other end of the spectrum, high LSA-dimension and pure truncation provided very little overlap of keywords and also resulted in small values of C. For low LSA-dimensions marginally larger than the dimension achieving saturation mentioned above, the values of C peaked, resulting in very large lattices for the example collection. However, for LSA-dimensions around 300, we have found the number of concepts to be very near linear (in the number documents) for larger collections spanning a limited domain, provided that the number of keywords assigned to each document does not exceed 15-20 (see appendix J.7.5).

The space requirements of the algorithm is near-linear in the number of concepts C. “Near” because the size of each concept slowly grows as the average number of children and parents increases when the lattice grows. The average number of documents connected with each node (the average extent) might also grow as the number of documents increases. Combining this with the above empirical value of C we get a space complexity somewhere between linear and quadratic, depending on various properties of the domain (conceptual span, size etc.) and the applied keyword extraction scheme (number of keywords per document, overlap of keywords, LSA-dimension etc.).

The computational complexity of adding a single document theoretically lies somewhere between linear and quadratic in the number of concepts C, depending on how many times `link` is invoked. However, experiments carried out in [CR96] surprisingly show that in practise, the run-time is very near linear in C. The number of calls to `link` never exceeds a small fraction of the concepts examined. Combined with the empirical results stating that C is almost linear in the number of documents. This is expected, since only an almost fixed (slowly growing) number of new concepts is added per document – all concepts are thus only examined a "pseudo constant "number of times independent from C when adding a document.

So, if we consider the generation of the *whole* lattice, the computational complexity can thus be estimated to be between quadratic and cubic in the number of documents N (since the cost of adding a singe document turns out to be very near linear in C). However, our own empirical results, showing almost linear growth in lattice size, point towards an actual empirical complexity close to $O(N^2)$ (see section 9.6). For large collections the saturation effect discussed above might[20] reduce this to almost linear (with a large constant represented by the almost fixed number of concepts in the lattice).

7.4.4 Advantages

Since GALOIS is incremental, the lattice (clustering) can be updated as documents are added to the collection[21]. This is a great advantage in connection with websites since they are often updated with new content, in some cases every hour, and it would thus be an advantage if the clustering could be kept up to date easily without recalculation of the entire clustering.

Clustering into a Galois lattice has several interesting advantages compared with more traditional clustering methods (e.g. K-Means, PDDP etc.). A Galois lattice for instance, contains a natural description of the generated clusters (concepts) via the notion of intent. Labelling of clusters thus becomes much easier and represents a "correct" definition of the clusters' content (e.g. this cluster contains all documents sharing "philosophy", "logic" and"Aristotle" as common keywords). The generated concepts (clusters) are in addition completely soft – a document will be assigned to all concepts having an intent that is a subset of the document's keywords. It is as a result possible for a document to appear in several unrelated clusters (e.g. a document containing information on Chinese medicine might both appear in clusters related to alternative medicine and in clusters related to Chinese culture and tradition).

Finally, as was already discussed above, the lattice structure represents some advantages compared with ordinary tree-based hierarchical structures when used to organise knowledge in a domain. The dual algebraic definition of a lattice further allows interesting applications, such as using algebraic operations to induce the least general concept (cluster) that covers a collection of documents and hence create a kind of a multi-document "more-like-these".

Compared with Apriori, which we will detail in the next section, GALOIS has the advantage that it creates the entire Galois lattice by forming all possible concepts implicitly present in the given document collection. Apriori on the other hand only creates the concepts that have a supporting extent of a specified minimum size.

[20] It is difficult to predict when (and whether at all) saturation will occur for a given collection.

[21] This is not possible using the LSA keyword extraction scheme, since it, in its present implementation, is dependent on global information to calculate the SVD. However, an incremental SVD-like decomposition is proposed in [Bra02] and [Bra03] that might be used to overcome this limitation of standard SVD algorithms.

7.4.5 Weaknesses

The greatest weakness of GALOIS is its uncertain computational complexity. Even with the empirically determined estimates found above, one never knows how well the algorithm will perform (speed and memory-wise) on a given collection. In addition, the algorithm carries (an estimated) best case computational complexity that is quadratic which is quite prohibitive for large collections. This might be amended somewhat by the discussed saturation effects, however, these effects might never materialise if the collection spans a large or several knowledge domains.

Another weakness, is that clustering approaches based on formal conceptual analysis rely on binary relations between objects and attributes. GALOIS thus doesn't take the weights of the keywords into account when clustering, it only works on the Boolean keyword space, considering the relation between documents and keywords as strictly binary. It might thus miss subtle differences between the documents which are only evident in the keyword weights (e.g. is a document concerning Chinese medicine mostly related to "China" or to "medicine", if it contains both of these terms as keywords? The keyword weights might reveal that the document is more about "medicine").

Using the Boolean keyword space also requires a larger overlap of the keywords to find meaningful clusters. Too small a degree of overlap will lead to a few, small and unrelated clusters. We thus need LSA to compress the keywords of similar documents together. The time and memory requirements of the algorithm are also very sensitive to the number of keywords assigned to each page. The number of concepts grows more or less exponentially when this number is increased. GALOIS thus heavily relies on the hypothesis that all documents in a collection can be adequately described (for clustering purposes at least) by a relatively small number of central core-terms/keywords assigned to each document.

7.4.6 Implementation Details

Lattice nodes (concepts) are contained in memory during the lattice building process to make the process fast. The nodes consist of 4 vectors/sets: Intent, extent, children and parents – the nodes are thus double-linked into the lattice structure. In addition, all nodes are stored in a single vector allowing easy iteration over the nodes in the main loop of the algorithm. We also store a top node ($\top$) with all "orphan" nodes as children. However the orphan nodes do not link to the top as parent, since this is redundant - if a node has no parents we know that top is parent of this node. We do not store a bottom node ($\bot$) since we usually search the lattice in top-down direction. However a bottom node could easily be added (with links to all childless nodes) if needed.

Finding the upper boundary set G and the lower boundary set U is among the more computationally demanding operations. In practise the 2 boundary sets are determined by only considering a subset of the whole lattice. As the new node (we are adding) is an intersection of an old concept and a new object, the concepts to which the new node should be linked must be more general than the new object, or more general than the old concept. We thus only consider the union between the ancestors of the node corresponding to the new object and the ancestors of the old concept (that was used to form the intersection that led to the introduction of the new node). However, when a new node is added to represent the intent of a new object, the entire lattice needs to be searched to determine G and U. [CR96] claims that it is sufficient to determine either U or G and then derive the other set from the already calculated set. However, it doesn't specify how this is done most efficiently. We tried several ways of doing this (via the lattice linking structure), but our testing showed that brute-force search in the lattice was in most cases marginally faster[22].

We have made extensive use of profiling to identify hot spots in the code, this has let to several significant optimisations, most notable:

- Flattening of recursions when calculating upper- and lower bounds for a new node. This approximately yielded a $3 - 4\times$ speedup for the entire algorithm.

- Use of hashing to verify that insertions in large sets of nodes do not violate the set invariant (a given node only appears once in the set). This was done to speed up the calculation of the set of nodes that needs to be considered when searching for G and U. We use open addressing (all nodes are kept in the same array) relying on double hashing[23] to handle collisions. This approximately yielded a 2× speedup for the entire algorithm.

- Use of ordered sets to represent the intents, allowing cross-merge and binary search[24] to speed up intersection and subset operations on intents. This approximately yielded a 1.5× speedup for the entire algorithm.

In our implementation a Galois lattice is saved to disk using 3 separate tables:

1. The *lattice core* table, which contains intent and extent for every node (except $\top$).

2. The *lattice link* table, which contains the parents and children of every node (including $\top$).

3. The *page-to-node* table, which for all pages contains the nodes that have the page as part of their extent. This table is redundant, and is included to make look-up of nodes related to one or more pages fast.

The reason for splitting the lattice into core and link tables is that when the lattice is loaded (after it has been created and saved), the link structure of the lattice (the lattice link table) can be kept in memory, while the lattice core table is kept on disk to save memory. In this way the costly operations of converting lattice IDs (saved on disk) to pointers (in memory) is done once and for all. In addition, the splitting of the lattice into core and link tables turned out to make lattice generation using Apriori (detailed below) easier.

[22] However, this might not hold for very large lattices.

[23] The 2 hash functions that we applied (`h1` and `h2`) are very fast and efficient. They are based on 32 bit mix functions developed by Thomas Wang and Robert Jenkins and can be inspected in Appendix E.

[24] Binary search is only applied if the size of the set is larger than an empirically determined value, the overhead of calling `bsearch` were found to be too expensive for smaller sets.

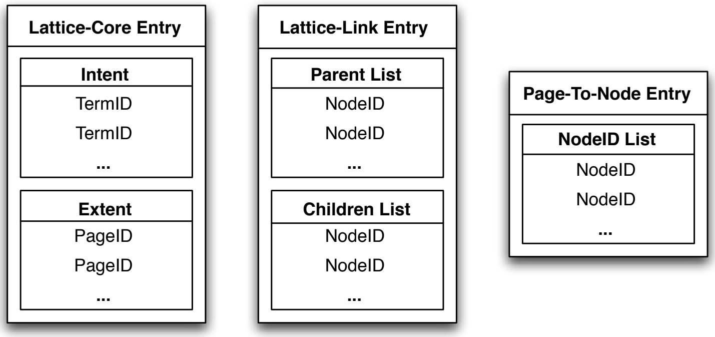


Figure 7.4: *The structures used to save a Galois lattice in the database.*

7.5 Apriori-Based Lattice Generation

7.5.1 Description

In this section we will outline and discuss an extended version of the popular *frequent itemset* mining algorithm Apriori that allows "pruned" Galois Lattices[25] to be formed much faster than the full Galois Lattices that can be generated with GALOIS. However, before we discuss the extensions that we have made to Apriori, we will start out with an introduction to the Apriori algorithm and the problem of finding *frequent itemsets.*

Background

Apriori and other approaches to frequent itemset mining originate from efforts to analyse and find useful patterns in customers' buying behaviour from large databases of customer transactions. Today, the automatic discovery of frequent itemsets is considered one of the most important areas in data mining, which is the emerging sub-field within statistics and pattern recognition, concerned with finding patterns and connections between elements in large databases [Bod03].

A *frequent itemset* is a set of items that often (more than *min_supp* times) occurs in the transactions in a given database. [Bak04] relays the following popular example of frequent itemset mining used to identify customer buying behaviour:

> "A popular example in the literature (possibly apocryphal) is processing the supermarket transactions of working men with young children: When they go to the store after work to pick up diapers, they tend to purchase beer at the same time. Thus, it makes sense statistically, if not socially responsibly, to put a beer refrigerator in the diaper aisle."

The problem of finding frequent itemsets starts with a database of transactions $\mathcal{T}$:

$$\mathcal{T} := \{t_1, t_2, ..., t_n\} \tag{7.10}$$

[25] See the previous section for a detailed explanation of Galois lattices and the theory behind them.

With each transaction t_i being an itemset ($t_i \subseteq \mathcal{I}$). The support of an itemset l ($\in \mathcal{I}$) in $\mathcal{T}$ is defined as the transactions that contain l as a subset and it is denoted in the following way:

$$supp_{\mathcal{T}}(l) = \{t_j \in \mathcal{T} : l \subseteq t_j\} \tag{7.11}$$

An itemset l is frequent if its support is greater than a given minimum support ($|supp_{\mathcal{T}}(l)| \geq min\_supp$). An itemset with k elements that is frequent is called a frequent k-itemset [Bod03].

The problem is thus to find all frequent itemsets in a given database of transactions $\mathcal{T}$. One of the most important contributions to solving this problem in an efficient way is the Apriori algorithm, which was proposed by Agrawal and Srikant in [AS94]. Apriori has quickly become the "gold standard" that all other frequent itemset algorithms are measured against and today the notion "Apriori" covers a whole family of algorithms based on the same basic ideas [Bod03]. However, the original Apriori algorithm will suffice for our purposes.

The Original Apriori Algorithm

In explaining Apriori we will utilise the following notation[26]:

- L_k is the set of frequent k-itemsets (those with min_supp support), each member is represented by the itemset and the support count.
- C_k is the set of candidate k-itemsets (potentially frequent itemsets), each member is also represented by the itemset and the support count.

In the following we assume that all itemsets are represented as ordered sets.

The Apriori algorithm is actually rather simple, Apriori exploits the basic fact that all subsets of a frequent itemset are also frequent. The initial pass directly finds the frequent 1-itemsets by counting items and finding the ones that are frequent (have min_supp or higher support). Subsequent passes (pass k) have 3 basic steps:

1. **Candidate generation** is the join step where L_{k-1} is joined with itself to generate the candidates in C_k, i.e. it is the feedback step were the frequent $(k-1)$-itemsets are used to generate possible candidates for the frequent k-itemsets. As can be seen in the algorithm below, we only need to pair up itemsets from L_{k-1} that differ in their final element (since the itemsets are assumed ordered). This candidate building process is from where the name Apriori derives – we use a priori information deduced from the last step to determine the candidates that need to be examined in the current step.
2. **Candidate pruning** check that all $k-1$ subsets of the candidates in C_k are frequent, else the candidate is removed from C_k.
3. **Candidate support** determines the support of the candidates in $\mathcal{T}$. All candidates with less than min_supp support are removed and not saved in L_k. This is the most data

[26] Adapted from [AS94]

intensive part of the algorithm since we need to iterate through all transactions in the database.

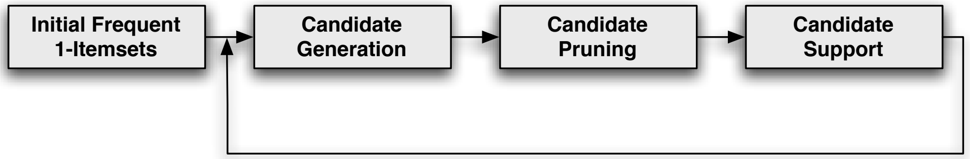


Figure 7.5: *Process flow of the original Apriori algorithm*

This progressive build-up is at the heart of Apriori and the feedback loop is repeated until no more frequent itemsets are found ($L_{k-1} = \emptyset$). The process flow of Apriori is illustrated on figure 7.5[27].

Our Extensions to Apriori

Our extensions to Apriori are inspired by [BEX02] that states:

> "Due to the monotonicity property, the frequent term sets form a lattice structure: All 1-subsets of frequent 2-sets are also frequent, all 2-subsets of frequent 3-sets are also frequent etc. This property can be exploited to discover a hierarchical frequent term-based clustering."

Before we explain the extension we have made to Apriori, we will first show how the above lattice of frequent itemsets is related to formal concepts and Galois lattices. Let us briefly recall the definition of *formal concept* from the previous section (page 77): A *formal concept* is defined as a pair (A, B). A is called the *extent* of the formal concept and B the *intent*. The *extent* (A) of the formal concept describes the subset of objects (documents) that belong to the concept, while the *intent* (B) describes the common attributes (keywords) that these objects share:

$$A := \{g \in G \mid \forall m \in B : (g, m) \in I\} \tag{7.12}$$

$$B := \{m \in M \mid \forall g \in A : (g, m) \in I\} \tag{7.13}$$

If we look at documents as transactions and the keywords assigned to these documents as items, we quickly realise that an itemset (of keywords) fulfils the first part (equation 7.12) of this definition, having the itemset itself (l) as *intent* (B) and the support of the itemset $(supp_{\mathcal{T}}(l))$ as *extent* (A)[28]. However, we do not fulfil the second part (equation 7.13) of the definition, since a superset of the itemset l might have the same documents as support as l. In this case the itemset l does not represent a concept in the formal sense, it is rather a kind of redundant (too general) *pseudo-concept*. However, for our purposes these extra pseudo-concepts do not really

[27] Adapted from [Bak04].

[28] The database $\mathcal{T}$ is in this case the formal context (the document collection).

pose a problem. But, if so desired, the pseudo-concepts could be removed during the below outlined *candidate linking* step.

We further realise that the set of all frequent itemsets, which do not form pseudo-concepts, is a subset of the concepts found with GALOIS. This subset consists of all the concepts from GALOIS that have a minimum extent size of *min_supp*. These concepts can thus be used to construct a "pruned" sublattice of the Galois lattice (found with GALOIS), since any subset of the nodes of a complete lattice also forms a (complete) lattice (the Galois lattice is always a complete lattice). So the lattice formed by the frequent itemsets is hence this "pruned" Galois lattice, possibly with the addition of some extra pseudo-concepts that are not concepts in the strict formal sense.

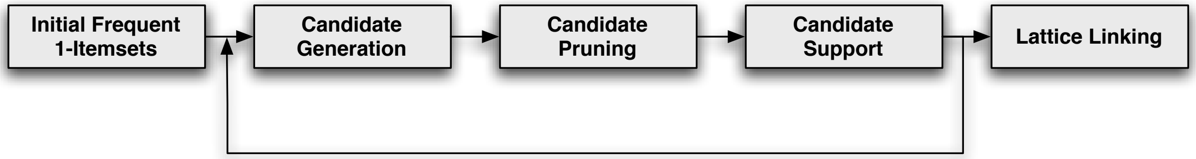


Figure 7.6: *Process flow of the extended Apriori algorithm*

We have extended the original Apriori algorithm so it returns the "pruned" Galois lattice discussed above (including additional pseudo-concepts). To do this we needed to make 2 changes to the algorithm:

1. First we had to save the transactions (documents) that support the itemset (as extent for the itemset) in the candidate support step. This was done by letting both L_k and C_k contain the extent in addition to the support count.

2. Secondly, we had to link the generated frequent itemsets into a lattice. This was done by adding a lattice linking step before terminating the algorithm (see figure 7.6). Each node (frequent itemset) is linked into the lattice by linking it to all nodes with an intent that is a $(k-1)$-subset of the node's intent. All frequent k-itemsets thus have k frequent $(k-1)$-itemsets as parents (with the exception of frequent 1-itemsets, that are linked to the top node).

The distance measure applied is thus the same as in connection with GALOIS; the inverse of the number of shared keywords. However, since the extended Apriori generates a "pruned" Galois lattice, there are no guarantees that 2 documents that share some keywords will be clustered together – it depends on whether the concepts that the 2 documents share have enough support to be included in the lattice.

7.5.2 Algorithm

Formally our extended version of the Apriori algorithm looks like this[29]:

[29] Adapted and extended from [AS94].

```
function apriori(transaction database $\mathcal{T}$, support threshold $min\_supp$)
        $L_1 = \{$frequent 1-itemsets in $\mathcal{T}\}$
        for $(k = 2; L_{k-1} \neq \emptyset; k{+}{+})$ do
            $C_k =$ candidate-generation($L_{k-1}$) //Generate and prune candidates
            for each transaction (document) $t \in \mathcal{T}$ do
                $C_t = \{l \in C_k \mid l \subseteq t\}$ //Candidates contained in $t$
                for each candidate $c \in C_t$ do
                    $c.count{+}{+}$
                    add $t$ to $c.extent$
            $L_k = \{c \in C_k \mid c.count \geq min\_supp\}$
        return lattice-link($\bigcup_k L_k$) //Link frequent itemsets into a lattice
```

```
function candidate-generation(frequent $(k-1)$-itemsets $L_{k-1}$)
        //Begin SQL notation
        insert into $C_k$
        select $p.item_1, p.item_2, ..., p.item_{k-1}, q.item_{k-1}$
        from $L_{k-1}\, p, L_{k-1}\, q$
        where $p.item_1 = q.item_1, ..., p.item_{k-2} = q.item_{k-2}, p.item_{k-1} < q.item_{k-1}$
        //End SQL notation
        return candidate-pruning($C_k$, $L_{k-1}$) //Prune candidates
```

```
function candidate-pruning(candidates $C_k$, frequent $(k-1)$-itemsets $L_{k-1}$)
        for each candidate $c \in C_k$ do
            for each $(k-1)$-subset $s$ of $c$ do
                if $(s \notin L_{k-1})$ then
                    delete $c$ from $C_k$
                    break
        return $C_k$
```

```
function lattice-link(frequent itemsets $L$)
        initialise $lattice$
        add empty top-element to $lattice$
        for each frequent itemset $l \in L$ do
            $k = |l|$
            for each $(k-1)$-subset $s$ of $l$ do
                find frequent itemset $l_p = s$
                add $l$ to $l_p.children$
                add $l_p$ to $l.parents$
                add $l$ to $lattice$
        return $lattice$
```

7.5.3 Time and Memory Complexity

The basic idea behind using Apriori to generate a "pruned" Galois lattice is that the support threshold (*min_supp*) will significantly limit the number of generated concepts and thus result in better performance both speed- and memory-wise compared to GALOIS.

The theoretical computational complexity of Apriori is (as with GALOIS) very high and difficult to estimate, since it depends on the number of found frequent itemsets in each pass[30]. Analogous to GALOIS, these numbers in turn depend on the number of documents and the size of the object description space (i.e. number of keywords per document, total number of unique keywords assigned to the documents etc.). However, Apriori is designed for doing data mining on very large databases and should in practise perform well on large document collections. Our own tests seem to confirm this (see section 9.7).

The linking step that we have added at the end of the algorithm has a computational complexity of $O(Ck)$, C being the total number of frequent itemsets found and k being the maximum size (number of items) of the frequent itemsets. This is because a frequent k-itemset always has k $(k-1)$-subsets that need to be linked to (as parents).

The memory complexity of our extended Apriori algorithm is in theory[31] only determined by the space needed to hold C_k and L_{k-1} in memory, since L_{k-1} can be saved on disk as soon as it has been used to generate C_k. However, C_k might actually require a considerable amount of memory since the number of candidates often far exceeds the number of frequent itemsets, and since we need to store the (possibly large) extent of each candidate in the candidate support step. However, Apriori is still expected to have lower memory requirements than GALOIS, which needs to store the entire lattice in memory (including the extents) during the lattice building process.

7.5.4 Advantages

As touched upon above, the main advantage of Apriori compared with GALOIS is that Apriori is expected to perform much better speed- and memory-wise on large document collections. This is a significant advantage since our tests show that GALOIS performs rather badly on large collections (see section 9.6). Additionally, Apriori more or less inherits the advantages of using a Galois lattice that was outlined in section 7.4.4 in connection with GALOIS. Apriori thus represents a computationally faster and more feasible way of gaining these advantages.

7.5.5 Weaknesses

The main weakness of Apriori compared with GALOIS is, on the other hand, that the returned Galois lattice is “pruned”. Itemsets with low support are left out, resulting in loss of the most specific concepts that only encompass very few documents. This loss of small concepts can have a negative effect on more-like-this and other applications that rely on specific clusters of few documents.

As was the case for GALOIS, the complexity of Apriori is still difficult to estimate since it depends on the conceptual span of the document collection. Apriori is in addition not an incremental algorithm, and the whole clustering process thus has to be repeated every time one wants to add new documents to the clustering.

[30] The time complexity of each pass depends on the number of candidates C_k and is thus dependent on the number of frequent itemsets found in the previous step (L_{k-1}).

[31] Our current implementation require somewhat more memory, since all found frequent itemsets are kept in a trie during all passes and since we build a *page-to-node* table in memory while finding the frequent itemsets.

Finally, the weaknesses of using conceptual clustering, which was discussed in section 7.4.5 in connection with GALOIS, also apply to Apriori. I.e. Apriori only uses the Boolean keyword space (it doesn't take the keywords ' weights into account) and also relies on sufficient overlap of keywords to achieve good results.

7.5.6 Implementation Details

Our implementation of Apriori is based on a fast trie-based C++ implementation by Ferenc Bodon[32]. We have extended this open source implementation to accommodate for lattice building as discussed above. The implementation uses ordered sets and a *trie*[33] to find support for candidate sets in a fast way. Experiments show[34] that this trie-based approach is faster than the hash-tree based approach originally proposed in [AS94].

Since all transactions in $\mathcal{T}$ are considered one by one in each pass (in the candidate support step), the implementation is designed to optionally store $\mathcal{T}$ in memory. This allows fast access to the transactions at the cost of increased memory consumption. To further speed up the algorithm, the generation and support determination of 2-itemsets are done via a simple vector approach, which for 2-itemsets is significantly faster than just using the trie (see [Bod03]). The pruning of C_k is done during generation of C_k to save memory (before a k-sized candidate is added to C_k, it is checked whether all its $(k-1)$-subsets are frequent).

As the algorithms progresses, the lattice is saved one "level" at a time (corresponding to the frequent $(k-1)$-itemsets). When a level has been saved to disk, the extent information of that level is deleted to save memory. The final lattice link stage is implemented as a depth-first tree traversal, finding and linking the parents to each node as outlined in the algorithm above.

The lattice is saved in the same 3 tables that were used in connection with GALOIS (i.e. *lattice core*, *lattice link* and *page-to-node*). The split of the lattice into the tables *lattice core* and *lattice link* makes it easy to save the core one pass at a time during the main algorithm, while deferring saving the linking information for each node until the final lattice link step.

Future Improvements

As briefly mentioned above, we currently build the *page-to-node* table directly in memory during the execution of the main loop of the algorithm. To save memory while running the main loop, the generation of the *page-to-node* table could be deferred to a later time. This could for instance happen just before the lattice link step.

In addition, we believe that the current implementation, while fast, is not optimal partly due to the overhead of using C++ and an object-oriented representation of the trie structure, and partly because the support for lattice generation is currently "bolted" on top of a standard Apriori implementation. A new custom made implementation in pure C (or FORTRAN), built from scratch to support lattice building might yield better performance than the current implementation.

[32] Available at `http://www.cs.bme.hu/~bodon/en/apriori/`

[33] Tries were discussed above in section 5.3.2.

[34] See discussion in [Bod03].

Chapter 8

Implemented Postprocessing

In this chapter, we will first detail the implementation of similar pages (more-like-this) for both of our cluster output formats; hierarchy-based and lattice-based. Thereafter, we will outline a preliminary approach for search result clustering, working on top of our lattice-based clustering algorithms. Finally, we briefly discuss the various small functions implemented for presentation and evaluation purposes.

8.1 Finding Similar Documents (More-Like-This)

As we mentioned in section 3.4, More-Like-This is an important application for clustering technologies, and we have chosen to implement this for both the hierarchy-based and lattice-based algorithms to test and demonstrate how this would work.

8.1.1 Hierarchy-based More-Like-This

K-Means, CURE and PDDP all operate in a term vector space as described earlier and are in our implementation all hard hierarchical clustering algorithms. Because of these similarities, we have stored the created cluster hierarchies in the same form in the database as described in the previous sections.

This makes it easy to create one common more-like-this function for all three algorithms. The general concept of this more-like-this function is to provide as many results as required (if possible). The results will be taken from the smallest (most specific) cluster first, then from the second smallest and upwards until enough results are provided. The results from each cluster are ranked according to their distance to the original document.

Algorithm

```
function MLT($DB, D, N$)
        ($DB$ is the database, $D$ is the original document and $N$ the number of results)
        $MLT\_result = \{\}$
        $clusters$ = get_clusters($DB, D$)
        for each cluster $C \in clusters$ do
            rank_and_sort_pages($C, D$)
            insert_with_rank($MLT\_results, C$)
        while $size(MLT\_result) < N$
        Return $MLT\_result$
```

The rank_and_sort_pages algorithm uses the cosine distance measure (using the documents' keywords, not their full term sets) and the ANSI C `qsort`. Other distance measures and sorting algorithms could easily be implemented. The rank_and_sort_pages algorithm produces a number representing the relevance[1] of the document relative to the original document.

8.1.2 Lattice-based More-Like-This

For the lattice-based algorithms, GALOIS and extended Apriori, similar pages (more-like-this) are calculated in much the same way as outlined above for the hierarchy-based algorithms.

First, all lattice nodes (clusters) containing the given page are located using the page-to-node table. These clusters are then ranked (with regard to the original page) and sorted using a ranking algorithm. This ranking is necessary, since the page might be contained in several lattice nodes on the same level (due to the soft nature of lattice-based clustering). The ranking algorithm compares the intent I of the given node with the assigned keywords K and corresponding weights W of the original page:

$$rank = \sum Q - E(Q) * \sum Q \tag{8.1}$$

Where E is the standard entropy function and Q the (set of) weights of the keywords that the document and the node intent share:

$$Q = \{w_j \in W \mid k_j \in (I \cap K)\} \tag{8.2}$$

This ranking algorithm fulfils 3 primary goals:

- For nodes with 1-word intents, it is the weights of that keyword in the document that determines the rank.
- Nodes with greater intent have a larger chance of getting a higher rank (depending on the sum of the weights of the keywords defining the node's intent and their weight distribution).

[1] Here, relevance simply refers to the cosine distance of the documents - 1 for most relevant, 0 for least.

- For 2 nodes having intents of equal length and equal (weight-)sum, the node with the most uniform weight distribution (in Q) will be ranked the highest due to the entropy function.

The rest of the lattice-based similar page algorithm works more or less like the hierarchy-based algorithm outlined earlier. The clusters/nodes are considered in descending rank order, all pages in the given cluster not already added are added ranked according to the cosine distance to the original document. This process repeats itself until the desired number of pages has been found.

In other words, the only difference between the two similar page algorithms is that the hierarchy-based algorithm simply considers the clusters according to size[2], whereas the lattice-based algorithm considers them according to a pre-calculated page vs. cluster rank.

8.1.3 Data Representation

Data Representation

The more-like-this functions that we have implemented are light-weight enough to run online for smaller databases, but for maximum performance and testing purposes, we have chosen to pre-calculate and store more-like-this results for all pages in a table. The common data structure used by both algorithms to store similar page information in the table is shown in figure 8.1.

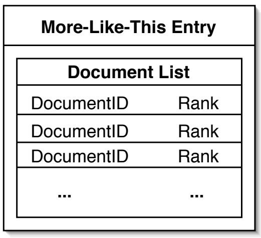


Figure 8.1: *The More-Like-This entry stored in the database.*

8.2 Search Result Clustering

We have implemented a preliminary search result clustering system to try to assess the feasibility of such a system when a clustering of the document collection is available. The preliminary approach is based on our lattice-based clustering algorithms[3]. The algorithm that we have implemented is designed to return a predefined number of categories (clusters) that cover the search result. This is optimised in such a way that the maximum number of documents from the search result are covered by the categories. The algorithm works like this:

1. Find all clusters/nodes containing documents from the search result (using the union of the page-to-node entries).

[2] This is an acceptable simplification since all the hierarchical algorithms create hard clusters (on each level).

[3] Due to time constraints, we have not been able to implement a search result clustering working with the hierarchy-based algorithms.

2. Add the node having the most pages from the result set as a category.
3. Build a list of pages in the result set that is not covered by the former category.
4. Add the not yet added node that has the largest intersection with the list (of pages not already covered).
5. Update the list of pages not already covered by the added categories.
6. Repeat from step 4 until the required number of “categories” has been found.
7. Finally, all pages not covered by a category are put in an artificial “misc.” category.

Before returning the result clustering, the added clusters can optionally be filtered to make their extents only contain documents from the search result. We have made this optional, since some applications of search result clustering might benefit from the full clusters, including documents not matched by the original query. E.g. the clusters best covering a result could be added as search results themselves, allowing the user to navigate into a cluster and see the content of the entire cluster.

We use the keywords of the node intents as labels for the found categories. In most cases, these labels work quite well in defining the common denominator for the pages in the cluster. However, in some cases, the connection is not immediately clear.

We have also briefly investigated a more elaborate approach, where the sublattice covered by the search result is returned. This could for instance be used as a basis for interactive navigation and browsing of search results (for instance see [CR95]). However, for larger search results, the sublattices involved proved to be rather large. This presented difficulties in determining how to present these lattices to the user in a simple and comprehensible way. At this preliminary stage, we therefore opted for the alternative approach as outlined above.

8.3 Evaluation and Presentation

We have implemented the presentation and evaluation utility functions as required in section 4.3.1 on page 46 to allow inspection and evaluation of the results produced by the various stages of the clustering toolkit. We have thus been able to use the HTML output to manually inspect and validate both keyword[4] and more-like-this tables[5], as well as the search result clustering[6], throughout the development process. In addition, we have also created small functions to inspect the clusters found by the various algorithms. However, the primary output of our clustering toolkit is the XML files detailing the similar pages for each page in the database, since these form the basis of the user test performed in chapter 11.

Since most parameters of our system concern preprocessing settings, we have implemented the ability to generate the following statistical information for a given keyword table:

- Number of unique keywords used (the dimension of the keyword space).

[4] See a sample of this in appendix H.
[5] See a sample of this in appendix I.
[6] See appendix K.

- Average number of keywords assigned to each document.
- The percentage of keywords contained in the documents they are assigned to. This is used in connection with Synergy, since this algorithm is able to infer keywords that are not present in the document (see chapter 6).

The above statistics can, together with the statistics showing the difference between two keyword tables, be used to assess the results of changing one or more parameters in the preprocessing and keyword extraction stage.

Chapter 9

Performance of the Implemented Algorithms

This section is concerned with testing the time and memory performance of our implemented algorithms. This is very important, since the scalability of the algorithms becomes critical as the size of collections increases beyond small websites. It is also interesting to see whether the theoretical complexities given for the algorithms is reflected in the actual running-time.

For the performance tests, we have chosen to use subsets of Wikipedia[1] ranging in size from 1,000 pages (documents) to 100,000 pages. This collection is somewhat atypical since the number of new terms introduced per document is expected to be higher than for collections only covering few or a single domain. Wikipedia is thus more or less "worst-case" regarding the growth of the number of terms compared with the number of pages. The test was carried out on a standard PC containing a 3 GHz Pentium 4 with 1 GB RAM and running Windows XP.

The test was performed using the settings given in appendix J.1. Important to note is that we do not extract bigrams, since the running-time and memory consumption of this step depends on the number of bigram candidates generated by one of our proposed schemes and not necessarily on the number of documents. For the interested reader, we can mention that the bigram extraction process takes approximately 10 minutes for 100,000 documents using scheme 2.

Furthermore, for these tests, we have chosen to aim for smallest clusters of approximately size 10. To do this, we have set the minimum support of Apriori to 10. For K-Means and PDDP, we have set the number of partitionings to 40 % of the number of documents, since our informal testing shows that this produces smallest clusters of 10 or less. For CURE and GALOIS, there is no way to limit the produced smallest clusters, since both algorithms are "agglomerative" and begin with a separate cluster for each document.

First, we will test the running-time and memory requirements of the entire preprocessing stage (excluding bigram extraction). Then we move on to testing the performance of the keyword extraction based on Synergy[2]. Hereafter, we test the running-time and memory requirements of the 5 implemented clustering algorithms. Finally, we have tested the construction of a similar page table for all pages based on an Apriori-based clustering[3]. We have not tested our search

[1] Wikipedia is an online encyclopedia, see `http://www.wikipedia.org/`

[2] We have not tested truncation-based extraction since this is very fast and scales near-linearly with the number of documents.

[3] We have not tested the similar page table construction connected with the hierarchy-based algorithm, since

result clustering, since our implementation is unoptimised and very preliminary[4].

9.1 Preprocessing

The preprocessing stage consists of several substages, including data extraction[5], filtering, stemming and weighting. As can be seen in appendix J.2.1, the running-time is roughly linear and ranges from approximately 9 seconds for 5,000 pages to approximately 200 seconds for our largest test collection of 100,000 pages.

Similarly, the memory consumption scales linearly and ranges from 95 MB at 5,000 pages to 283 MB at 100,000 pages as can be seen in appendix J.2.2.

The total number of terms is growing degressively as the number of documents in the collection increases (see appendix J.2.3). After stemming, the number of terms is reduced to a smaller number of stem groups, but the amount of stem groups is still growing (as also seen in the appendix). The reduction in terms after stemming is slowly decreasing from 34% at 5,000 pages to 25% at 100,000 pages as can be seen in appendix J.2.4. The size of the reduction decreases because the number of common words with several stems saturates earlier than special words (like proper nouns) that end up in their own stem groups.

Finally, appendix J.2.5 shows that the average number of terms per document declines as the collection grows. This ranges from 3 new words per document at 5,000 documents to below 1 new word per document for 100,000 documents. This development is natural since the most commonly used words are quickly saturated, and after this, only rare words and a few proper nouns are occasionally added. For algorithms with a complexity depending on the number of terms, this is for obvious reasons good. One should also note that Wikipedia as mentioned probably spans a much larger vocabulary than "normal" sites.

9.2 Synergy

The keyword extraction of Synergy takes place in two stages, first the Singular Value Decomposition of the term-document matrix is performed, then the actual keywords are extracted by constructing the approximated term-document matrix. For both the SVD part and the actual keyword extraction part of Synergy, the running-time and memory consumption can be inspected in appendices J.3.1 and J.3.2, respectively.

9.2.1 SVD

According to section 6.1.3, the complexity of the SVD library we use, should (for a given dimension) increase linearly with the number of non-zero elements (document-term relations) in

these algorithms (besides from an extra step in the lattice-based algorithms) are identical.

[4] For a collection size of 10,000 documents, the search result clustering is currently performed in "subsecond" time.

[5] This is not strictly preprocessing according to our architecture, but in the performance test, we have chosen to consider it as such.

the matrix. As can be seen in appendix J.3.3, these grow more or less linearly with the number of documents, ranging from 900,000 at 5,000 pages to almost 14 million at 100,000 pages. In full accordance with the above complexity, the running-time of the SVD algorithm scales more or less linearly with the number of documents, ranging from 35 seconds at 5,000 pages to 700 seconds at 100,000 pages.

With regard to memory consumption (for a given dimension), the theoretical complexity scales linearly with the number of documents plus the number of terms. Our testing shows that the memory consumption increases more or less linearly (slightly sublinearly) with the number of documents, which is also in accordance with the theory, since the number of terms (as outlined above) increases degressively with the number of documents in the tested range. Memory consumption ranges from 95 MB at 5,000 documents to 974 MB at 100,000 documents.

9.2.2 Extraction

As we discussed in section 6.1.3, the worst-case running-time complexity of the extraction stage is the number of terms times the number of documents for a given dimension. Our testing shows that the running-time grows in full accordance with this, ranging from 36 seconds at 5,000 documents to 4,338 seconds at 100,000 documents. When the number of floating point operations performed per second is calculated, it turns out that the algorithm performs the matrix multiplications at approximately 850 Mflops[6], which is not bad considering the different "bookkeeping tasks" that are performed to find the keywords for each document at the same time.

Theoretically, the memory consumption of this stage should (for a given dimension) also scale with the number of documents plus the number of terms. Our experiments more or less verify this, but due to the constants involved[7], the memory consumption of this stage is several times lower than that of the SVD stage. It ranges from 34 MB at 5,000 pages to 210 MB at 100,000 pages.

9.3 K-Means

Section 7.1.3 explains that the running-time of K-Means has a theoretical worst-case upper bound of the number of documents times the number of clusters times the maximal number of iterations. Since the maximum number of iterations is a constant and the number of clusters is preset to 80 % of the number of documents[8], the expected running-time should grow quadratic with the number of documents. As seen in appendix J.4.1, this is more or less the case. However, if we look at the average running time per document in appendix J.4.2, it is slightly degressive, which indicates a slightly subquadratic actual running-time. The running-time ranges from 10 seconds for 5,000 documents to 1,189 seconds for 100,000 documents.

[6] Millions of floating point operations per second.

[7] The reason that the memory consumption is approximately 4 times lower is that the extraction stage stores the values as `floats`, whereas the SVD stage stores the values as `doubles` and internally stores this twice in one of the algorithmical steps.

[8] The total number of clusters in the hierarchy is always twice the number of bisections, which we have predetermined to 40 % of the number of documents.

Assuming a relatively balanced bisection tree, bisecting K-Means is near-linear ($n\log n$) in terms of its memory consumption (see section 7.1.3). As seen in J.4.3, this is also more or less the case. The memory consumption ranges from 23 MB for 5,000 documents to 164 MB for 100,000 documents.

9.4 CURE

As described in section 7.2.3, CURE has a superquadratic worst-case running-time. However, as seen in appendix J.5.1, the algorithm actually scales quite close to quadratic[9]. Unfortunately, the constants involved are fairly large, resulting in the worst running-time of any of the algorithms. We had to stop testing after 20,000 documents, as the running-time in this case exceeded 3 hours. For 1,000 documents, CURE used 34 seconds, while it took 3,030 seconds (almost an hour!) for 10,000 documents.

As we stated in section 7.2.3, the theoretical memory complexity of CURE is linear. Our tests more or less confirm this as seen in appendix J.5.2, but because of our limited number of tests, this should be taken with a grain of salt.

9.5 PDDP

In section 7.3.3, we showed that the computational complexity of PDDP scales with the number of partitionings times the total number of non-zero elements in the keyword-document matrix. Since the number of keywords per document is limited by a predefined maximum, the complexity is quadratic in the number of documents, since the number of partitionings is predefined as 40% of the number of documents. The actual running-time of PDDP is shown in appendix J.6.1, where it ranges from 14 seconds at 5,000 pages to 1,624 seconds at 100,000 pages. As can be seen in appendix J.6.2, the average running-time per document is increasing linearly, which confirms that the algorithm runs in quadratic time.

As with the bisecting K-Means, the theoretical memory complexity of PDDP should be near-linear assuming a fairly well-balanced partitioning tree. In appendix J.6.3, we see that the memory consumption is somewhat linear, though unsteady. The memory consumption of PDDP ranges from 17 MB at 5,000 pages to 85 MB at 100,000 pages, making PDDP the algorithm that performs best with regard to memory consumption. The unsteady scaling is probably due to the memory requirements of the SVD algorithm that we use to calculate the principal direction that scales linearly in the number of documents plus the number of terms. This means that the memory consumption of the SVD part should grow degressively until the number of terms added per document has stabilised.

9.6 GALOIS

In section 7.4.3, we estimated the running-time of GALOIS to scale somewhere between quadratic and cubic – hopefully closer to quadratic. The actual tests confirm that the running-time

[9] To emphasise this, we have added a quadratic function closely matching the running-time.

is near-quadratic as seen in appendix J.7.1[10]. However, the constants involved are fairly large, resulting in the second worst running-time of any of the clustering algorithms. We stopped testing after 30,000 documents, since the running-time in this case exceeded 1 hour. For 1,000 documents, GALOIS used only 2 seconds, while it took 1,914 seconds for 20,000 documents.

Assuming linear growth in lattice size, we estimated the memory requirements of GALOIS to be near-linear, which is confirmed by our experiments in appendix J.7.2. The memory consumption of GALOIS ranges from 15 MB at 1,000 documents to 200 MB at 20,000 documents, making GALOIS the most memory-hungry algorithm. This is due to the lattice size that (for larger collections) grows linearly with the number of documents (see appendix J.7.3). The lattice size ranges from approximately 10,000 nodes at 1,000 pages to approximately 350,000 nodes at 20,000 pages.

The growth in lattice size is more easily visualised in the plot as seen in appendix J.7.4, showing the average and differential growth in lattice size for each added document. The plot clearly shows that, for larger collections, the number of new nodes per added document seems to stabilise around 20. This linearity can probably be attributed to the use of Wikipedia as document collection, since it is expected that Wikipedia articles should introduce new concepts at a steady rate.

For a collection spanning a single knowledge domain, saturation might be achieved. This can be seen in appendix J.7.5, where we have performed GALOIS on the CISI collection of information science abstracts. For higher LSA-dimensions (150 and 600), the number of nodes added per new document stabilises much earlier than was the case with Wikipedia. For a very low number of LSA dimensions (5), we see a saturation effect already occurring around 400 documents.

In general, the lattice depends greatly on the LSA-dimension used in the keyword extractor (see appendix J.7.6). For very low dimensions, saturation effects occur (as discussed above). For high dimensions, the overlap between keywords decreases resulting in only few and small lattice nodes (most documents are orthogonal). For moderately low dimensions, saturation no longer occurs and we have maximum overlap resulting in maximum lattice size.

Another important factor determining lattice size is the average number of keywords assigned per document. The lattice size grows very progressively as seen in appendix J.7.7. This severely limits the average number of keywords that can be used with GALOIS without running into a combinational explosion of lattice nodes.

9.7 Apriori-Based Lattice Generation

As discussed in section 7.5.3, the running-time of Apriori is expected to be lower than that of GALOIS. As can be seen in appendix J.8.1, Apriori scales near-linearly for large collections, ranging from 2 seconds at 5,000 documents to 226 seconds at 100,000, making it the fastest of the clustering algorithms tested. If we look at the average running-time per document (see appendix J.8.3), it more or less degressively increases slightly over the range.

As our current implemenation builds the page-to-node table in memory, it was expected that the algorithm would consume quite a lot of memory and scale near-linearly with regard to memory.

[10] To emphasise this, we have added a quadratic function almost matching the running-time.

In appendix J.8.2, the results confirm this, where memory consumption is seen to range from 21 MB at 5,000 pages to 625 MB at 100,000 pages. If we look at the average curve for memory consumption, for large collections, it seems to increase degressively.

With regard to the size of the produced lattice, it was expected to grow more or less in the same fashion as GALOIS, albeit with smaller constants. Our results in appendicies J.8.4 and J.8.5 confirm this, showing near-linear growth in lattice size for large collections. The size ranges from approximately 20,000 nodes at 5,000 pages to approximately 1.3 million nodes at 100,000 pages. The average amount of nodes added per document seems to stabilise around 13.

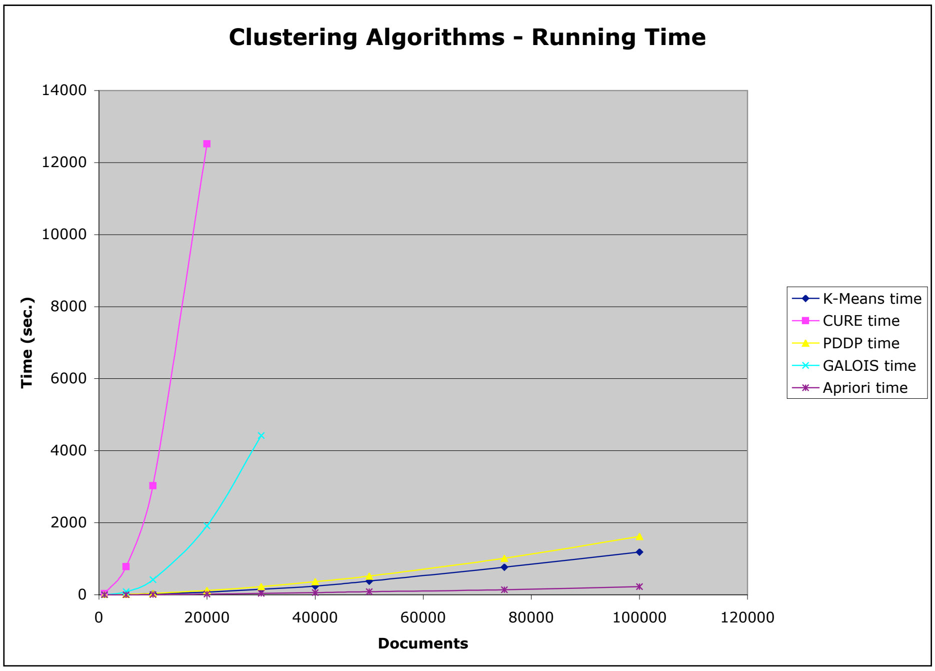


Figure 9.1: *The running-time of the clustering algorithms.*

9.8 Postprocessing: Finding Similar Pages

The time needed for constructing an Apriori-based similar page table containing entries for all pages can be inspected in appendix J.9.1, where it grows more or less quadratically. This is because similar pages are found for every page, and the time of this step is depending on the number of cluster that the given page is in, which again depends on the number of documents in the entire collection. This is also confirmed in appendix J.9.2 showing the average running-time per page. The running-time ranges from 1 second for 5,000 documents to 179 seconds for 100,000 documents.

The memory consumption scales near-linearly due to caching of the underlying lattice tables (see appendix J.9.3). The memory consumption ranges from 17 MB at 5,000 pages to 447 MB at 100,000 pages.

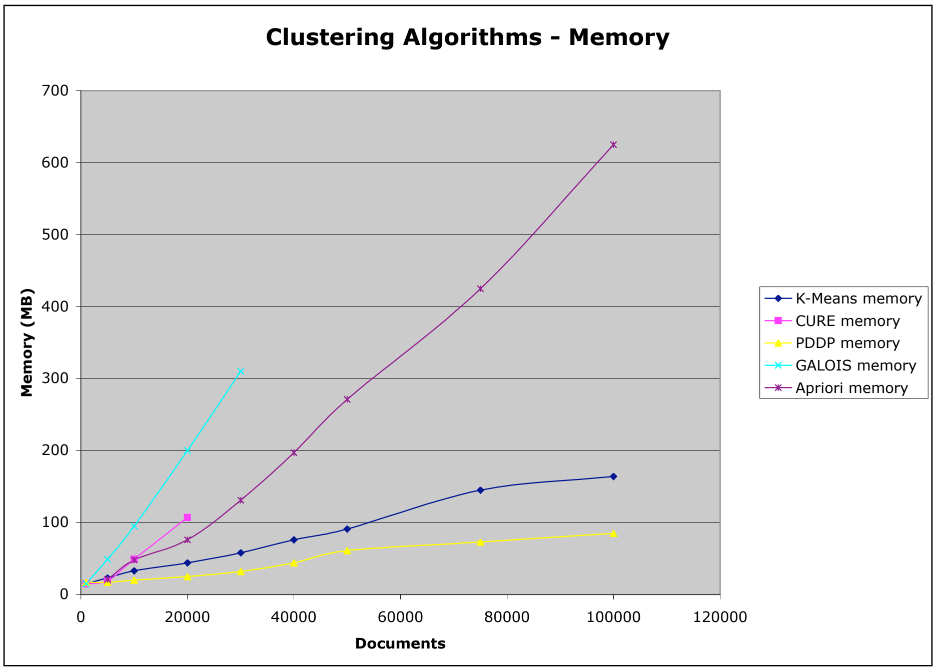


Figure 9.2: *The memory requirements of the clustering algorithms.*

9.9 Final Remarks

We have summed up the running-time of the clustering algorithms in figure 9.1 and the memory consumption in figure 9.2. As can be clearly seen, GALOIS and CURE perform (especially with regard to running-time) much worse than the other algorithms, and they are thus not feasible in connection with large collections.

Both K-Means and PDDP scale quadratically with regard to running-time, since we increase the number of partitionings with the size of the document collection. However, the constants involved are rather small, and both algorithms are therefore feasible for collections of at least 100,000 documents. In addition, for both PDDP and K-Means, the quadratic complexity can be transformed into linear complexity, if one is satisfied with a fixed number of partitionings (and thus, clusters). Both algorithms scale excellently with regard to memory consumption.

Apriori performs near-linearly for large collections (and with low constants). However, memory consumption might be an issue for large collections, but this might be improved if the page-to-node table is not built directly in memory. In addition, Apriori can be made to perform even better (both with regard to speed and memory), if a percentage of the document collection size is used as minimum support instead of a fixed number of documents.

As a conclusion, to the tests we have performed, we have decided to continue investigating K-Means, PDDP and Apriori. As for CURE and GALOIS, these have clearly shown that they (at least in their current implementation) are not feasible for the sizes of websites Mondosoft would like to apply clustering to.

Chapter 10

Toolkit Evaluation

In this chapter, we outline some of the experiences that we have gained while developing and using the clustering toolkit. First, we discuss some of the challenges that we have found in connection with clustering webpages as opposed to pure text. Hereafter, we make a sensitivity analysis showing the extracted keywords' sensitivity to various algorithm parameters. Finally, we provide 4 cases demonstrating how our preliminary search result clustering works in connection with searching for ambiguous terms.

10.1 Difficulty of Clustering Web Pages

During our work with this project, we have realised that clustering webpages is connected with several challenges and things to be aware of compared with clustering web pages and pure text documents. These include:

- Pages with mixed content, for instance news pages where the content of different subsections of the page is unrelated. This presents a serious problem with regard to clustering, since some subsections might be related to one set of pages and some subsections to others. One of the worst examples of this is news archive pages, where the diversity of different news entries might make the page similar to almost every other page on the site.
- Headers and navigation menus repeated on each page also present a challenge, since the menus might include important keywords from the domain, which are thus present on every page. The dilemma is that these words will either be so ubiquitous that the global weighting scheme will more or less dilute their importance on pages where the words are important, or the word will be present on just enough pages to make these pages cluster around the word.
- Pages with low textual content (e.g. image galleries), are another problematic area. Here, content present on all pages (see above) becomes an even larger problem, since the signal-to-noise ratio will be low – i.e. the words related with the page contents are suppressed by the relatively larger number of unrelated words on the page.
- Another challenge specific to websites is navigational pages ("index" and portal pages) that often do not contain relevant content usable for clustering.

- Alternating unrelated content (such as “recent news”, “tip-of-the-day” and polls) is often part of the layout for pages. This confuses clustering algorithms since the main topic of the page becomes obfuscated.

- Pages, where text is saved as images for layout purposes obviously present yet another problem since this text is not indexed.

10.2 Sensitivity Analysis of Algorithm Parameters

In this section, we have attempted to establish the sensitivity of the various input parameters to the preprocessing and keyword extraction modules. This is done as a series of 1-dimensional tests, where only one parameter is changed and everything else is kept equal. We have used the default settings outlined in appendix F as basis for the sensitivity analysis[1]. The test is performed using a 5,000-page subset of Wikipedia[2].

The result of changing a single parameter is evaluated through the various statistics generated for the keyword entries. We also provide the percentage of common keywords that remain after changing the variable. However, this statistic does not show whether the keywords assigned to a given document have changed their weights and as a result, their mutual order.

10.2.1 Minimum Document Frequency

The below table shows the sensitivity when changing the minimum document frequency[3]:

Minimum DF	**Default (10)**	**1**	**20**
Terms	10,484	97,360	6,269
Non-zero Elements	918,185	1,116,953	840,012
Assigned Keywords	55,777	56,690	55,130
Average Keywords per Page	11.16	11.34	11.03
Unique Keywords	2,276	2,657	1,889
Bigram Keywords	5.2%	7.7%	2.9%
Keywords from Page	84.1%	79.5%	86.9%
Common Keywords	-	76.0%	81.0%

The first thing to notice is that the number of terms almost increases by an order of magnitude when a minimum document frequency of 1 is specified. As a result, the (non-SVD part of the) keyword extraction process will run almost 10 times slower. As for the bigrams, we see that a low minimum frequency leads to relatively more bigram keywords, while a high minimum frequency reduces the relative number of bigrams in the collection. This might be attributed to the “rarity” of bigrams compared with normal terms.

[1] Please see section 11.2 below for an explanation of why these settings were selected as defaults.

[2] http://www.wikipedia.org/

[3] For both normal terms and bigrams.

Overall, the keyword extraction algorithm is somewhat sensitive to the minimum frequency, since only approximately 80% of the original keywords remain after changing the settings in either direction.

10.2.2 Maximum Document Frequency

The below table shows the sensitivity when changing the maximum document frequency:

Maximum DF	**Default (50%)**	**25%**	**100%**
Terms	10,484	10,469	10,488
Non-zero Elements	918,185	883,332	950,749
Assigned Keywords	55,777	55,570	55,776
Average Keywords per Page	11.16	11.11	11.17
Unique Keywords	2,276	2,281	2,279
Bigram Keywords	5.2%	5.3%	5.2%
Keywords from Page	84.1%	83.5%	84.0%
Common Keywords	-	95.2%	99.7%

The above table shows that, in general, the keyword extraction algorithm is not that sensitive to the maximum document frequency. This is probably because the log-entropy weighting function already downscales the weight of frequent terms. However, having a maximum frequency threshold decreases the number of non-zero elements slightly, leading to better performance in the keyword extraction stage.

10.2.3 Filtering 1-Character Terms

The below table shows the sensitivity when disabling 1-character term filtering:

1-Character Filtering	**Default (ON)**	**OFF**
Terms	10,484	10,512
Non-zero Elements	918,185	928,040
Assigned Keywords	55,777	55,832
Average Keywords per Page	11.16	11.17
Unique Keywords	2,276	2,266
Bigram Keywords	5.2%	5.1%
Keywords from Page	84.1%	84.3%
Common Keywords	-	94.2%

Overall, the keyword extraction algorithm is not that sensitive to 1-character term filtering, since only a limited amount of 1-character terms are possible (all alpha-numeric characters). However, 5% of the keywords change when the filtering is disabled, signifying that at least some 1-character terms are assigned as keywords. This is not desirable, since 1-character terms do not carry much meaning (if any).

10.2.4 Using Bigrams

The below table shows the sensitivity when disabling bigrams:

Find Bigrams	Default (ON)	OFF
Terms	10,484	10,077
Non-zero Elements	918,185	902,836
Assigned Keywords	55,777	55,727
Average Keywords per Page	11.16	11.15
Unique Keywords	2,276	2,252
Keywords from Page	84.1%	85.6%
Common Keywords	-	81.59%

The above shows that even though bigrams only account for 5.1% of the assigned keywords with the default setting, almost 20% of the assigned keywords change when bigrams are disabled. This is probably due to the way that SVD works, and we must thus conclude that the keyword extraction algorithm is sensitive (to whether bigrams are used or not) beyond the actual bigram keywords.

10.2.5 Local Weighting

The below table shows the sensitivity when changing the local weighting function:

Local Weighting Function	Default (Mondosoft's Weight)	Term Frequency
Assigned Keywords	55,777	56,301
Average Keywords per Page	11.16	11.26
Unique Keywords	2,276	2,291
Bigram Keywords	5.2%	5.1%
Keywords from Page	84.1%	87.3%
Common Keywords	-	81.33%

The keyword extraction algorithm is sensitive to the chosen local weighting function, since only around 80% of the keywords are the same after changing the local weighting function to term frequency. However, the sensitivity is not as high as one could have imagined, taking the difference between the two functions into account.

10.2.6 Global Weighting

The below table shows the sensitivity when changing the global weight function:

Global Weight Function	Log-Entropy	One	Log-IDF	Entropy	IDF
Assigned Keywords	55,777	55,530	56,892	54,579	61,810
Average Keywords per Page	11.16	11.11	11.38	10.91	12.36
Unique Keywords	2,276	1,359	2,337	952	3,886
Bigram Keywords	5.2%	0.1%	0.1%	6.8%	0.7%
Keywords from Page	84.1%	96.3%	86.8%	98.2%	46.1%
Common Keywords	-	52.2%	78.5%	36.6%	16.9%

The above table clearly shows that the keyword extraction algorithm is very sensitive to the applied global weight function. IDF, Log-IDF and the unit function (One) all significantly lower the relative number of bigrams assigned as keywords. This is evidence that the found bigrams have a different distribution in the document collection than "average" terms. As approximately 4% of the found terms are bigrams, the entropy-based algorithms apparently tend to favour this kind of distribution, while the other algorithms tend to do the opposite.

IDF appears to assigning more assigned keywords per page than the other algorithms. Furthermore, IDF also tends to favour inferred keywords (not in the page they are assigned to), whereas entropy and the unit function (One) seem to favour keywords present in the given page. However, the latter two functions also use a significantly lower number of keywords to describe the entire document collection, whereas IDF uses significantly more unique keywords to describe the collection.

Overall, the global weight function turns out to be the parameter that the keyword extraction algorithm is the most sensitive to. Surprisingly, IDF and Log-Entropy are the most different functions (they only have 16.9% of the keywords in common) even though both aim to promote rare terms.

10.2.7 Synergy: LSA-Dimension

The below table shows the sensitivity when changing the LSA dimension:

LSA Dimension	Default (300)	50	150	450	600
Assigned Keywords	55,777	61,825	57,078	55,336	55,003
Average Keywords per Page	11.16	12.36	11.41	11.07	11.00
Unique Keywords	2,276	602	1,360	3,110	3,729
Bigram Keywords	5.2%	2.3%	4.4%	5.4%	5.5%
Keywords from Page	84.1%	60.0%	75.8%	87.8%	90.1%
Common Keywords	-	36.2%	65.2%	77.9%	67.2%

As expected, the keyword extraction algorithm is very sensitive to the LSA-dimension, as can also be seen in appendix A. It is clearly seen that the number of unique keywords used to describe the collection increases as the size of the LSA-dimension increases, which is of course expected. Relatively, the number of bigram keywords also seems to increase as the LSA-dimension increases. Finally, the number of inferred keywords increases as we lower the LSA-dimension in full accordance with our experiments in appendix A.

10.3 Search Result Clustering

We have designed 4 small cases demonstrating the capabilities of the implemented preliminary search result clustering. Each case is designed as a 1-word search, where the search word has two or more meanings. An example of this kind of search could be "jaguar", where one might both get results related to the animal and the vehicle of the same name.

Each case is focused on the information needs of a hypothetical user and a corresponding 1-word search, where the word is ambiguous. As test data, we have used the same 10,000-page subset of Wikipedia, which is also used by our user test (see chapter 11). We have used a clustering that was made with Apriori using the settings in appendix F. We have chosen the following 4 search words as a basis of our demonstration:

- **Bush** could either be related to persons with this name (e.g. the two American presidents etc.), a specific class of plants (e.g. sagebrush) or a general concept referring to an area in nature (e.g. the "Australian Bush"). In this case, we have chosen the latter two meanings as the areas, the user is interested in, since both are related to nature.
- **Cd** is a common abbreviation used for several things including "Compact Discs", "Candela" and "Cadmium". In this case, we have chosen "Cadmium" as the user's area of interest.
- **Cluster** could either be related to astronomical clusters of stars and galaxies, clusters within biology, clusters as in this report or clusters of interconnected computers used in the area of high-performance computing. In this case, we have chosen high-performance computing as the user's area of interest.
- **Dwarf** could either refer to a biological definition of reduced growth or to astronomical objects like "red dwarfs". In this case, we have chosen astronomy as the user's area of interest.

For each case, we assess the results by locating the result clusters that are affiliated with the user's area of interest. We then calculate precision for each located cluster defined as the number of relevant documents compared to the total number of documents in the cluster. We thereafter calculate the combined recall of the clusters defined as the number of relevant documents found in these clusters compared with the total number of relevant documents in the search result.

For each case, we have gone through the entire search result and identified the documents that were relevant to the area of interest based on the way the search word was used in the document. The result of this is a list of relevant and irrelevant pages with duplicates[4] removed.

10.3.1 Case 1: Searching for "Bush"

In this case, a 1-word search for documents containing the word "Bush" yields 124 pages. We have applied our preliminary result clustering on these pages in appendix K.1[5], where we also

[4] Duplicate pages exist due to the structure of Wikipedia, which contains many pages that are identical except from their URL.

[5] The pages in the clusters are not ranked and in random order.

show the compiled list of relevant and irrelevant pages in the search result. As mentioned above, the user's area of interest was nature, thus the only cluster with an interesting label is `TREE`.

`TREE` contains 7 pages, all part of the relevant set. The precision is thus 100%. The size of the relevant set is 10 pages and the recall is thus 70%.

10.3.2 Case 2: Searching for "Cd"

In this case, a 1-word search for documents containing the word "Cd" yields 68 pages. We have applied our preliminary result clustering on these pages in appendix K.2, where we also show the compiled list of relevant and irrelevant pages in the search result. As mentioned above, the user's area of interest was chemistry, thus the clusters with interesting labels are `TABLE`, `METAL` and `ATOMIC`.

`TABLE` contains 17 pages, where 16 are part of the relevant set. The precision is thus 94.1%.

`METAL` contains 14 pages, where 13 are part of the relevant set. The precision is thus 92.9%.

`ATOMIC` contains 5 pages, where 5 are part of the relevant set. The precision is thus 100%.

The size of the relevant set is 16 pages that are all covered by the above clusters. The recall is thus 100%.

10.3.3 Case 3: Searching for "Cluster"

In this case, a 1-word search for documents containing the word "Cluster" yields 171 pages. We have applied our preliminary result clustering on these pages in appendix K.3, where we also show the compiled list of relevant and irrelevant pages in the search result. As mentioned above, the user's area of interest was high-performance computing, thus the only cluster with an interesting label is `CPU COMPUTER MACHINE`.

`CPU COMPUTER MACHINE` contains 5 pages, all part of the relevant set. The precision is thus 100%. The size of the relevant set is 5 pages and thus the recall is also 100%.

10.3.4 Case 4: Searching for "Dwarf"

In this case, a 1-word search for documents containing the word "Dwarf" yields 98 pages. We have applied our preliminary result clustering on these pages in appendix K.4, where we also show the compiled list of relevant and irrelevant pages in the search result. As mentioned above, the user's area of interest was astronomy, thus the clusters with interesting labels are `WHITE_DWARF` (as a bigram) and `STAR`.

`WHITE_DWARF` contains 32 pages, where 28 are part of the relevant set. The precision is thus 87.5%.

`STAR` contains 31 pages, where 29 are part of the relevant set. The precision is thus 93.5%.

The size of the relevant set is 36 pages and 33 of these are covered by the above clusters. The recall is thus 91.7%.

10.3.5 Final Remarks

In several of these cases, finding the relevant pages required browsing through almost the entire search result. However with the search result clustering, the excellent recall and precision numbers gained above is attainable by simply browsing a few clusters.

The above results indeed look very promising, both with regard to recall and precision. However, in general, one has to expect that search result clustering might improve precision at the cost of recall, since relevant documents might sometimes treat several areas and thus not end in the right clusters.

It is also important to emphasise that the above results are based on words with two or more separate meanings, and they are thus the ideal cases for using search result clustering. For normal searches, relevant and irrelevant pages might be closer to each other and thus harder to distinguish for the clustering algorithms.

Chapter 11

User Test

In this chapter we evaluate the quality of the similar pages found using the implemented clustering algorithms by means of a user test. The overall purpose of this user test is to establish how our algorithms compare to each other and how well they compare to a simpler search-based similar page approach. Comparing the algorithms based on their ability to find similar pages is also a good measure of how they compare based on the homogeneity of small clusters[1].

To assess the feasibility of using clustering in general – and our algorithms in particular – for finding similar pages, we have decided to test the algorithms against a simple, but effective scheme based on search (that Mondosoft has also been considering). The scheme works by performing a search for the 20 terms with the highest local weight in the given page. The idea is that pages that rank high in the result of this search must be more or less similar to the original page. Provided that the 20 query terms represent the content of the original page in a sufficient way, this "truncation plus search" scheme actually makes quite a lot of sense. We have thus chosen it as a baseline that can be used to assess whether our implemented clustering algorithms (and clustering in general) produce comparable results.

In addition to testing the implemented clustering algorithms, we would also like to assess the feasibility of our novel approach to keyword extraction (Synergy). More specifically, we would like to evaluate whether (when used for clustering) it produces small clusters of higher quality than those that can be obtained using truncation purely based on MondoSearch's local weighting (i.e. no filtering, no bigrams and no global weighting[2]). For larger clusters, our preliminary testing have determined that using truncation only leads to few and small clusters, due to the low overlap between keywords. While these small clusters might still be good for more-like-this, they would be disastrous in connection with search result clustering, since they are not large enough to cover a search result in a small number of large clusters – most search results would hence end in the "misc." category.

Our test thus has three objectives:

- Our main objective is to compare the quality of found similar pages for the three feasible clustering algorithms (K-Means, PDDP and Apriori).

[1] Due to the way we have implemented the similar page algorithm, it mainly utilises small clusters in the bottom part of the hierarchy/lattice.

[2] For technical reasons stemming could not easily be turned off, so stemming was still performed.

- Our secondary objective is to compare the clustering algorithms with the search-based similar page approach outlined above.
- Our tertiary objective is to compare Synergy's and the simpler truncation approach's suitability for subsequent clustering.

We have chosen to test the above based on the opinions of users, since a truly objective measure of the quality of found similar pages (and thus the underlying clustering) is difficult to provide. In addition, we believe that it is the users' (subjective) opinions and experiences that count, since humans after all are the ones that determine the success of any information retrieval system.

Below we will outline the test methodology and explain the choices of algorithm parameters, before finally presenting the results of the user test.

11.1 Methodology

11.1.1 Choice of Test Data

We have chosen a random 10,000 page subset of Wikipedia as our test collection, since Wikipedia articles usually have one easily grasped main topic and, in addition, they provide an interesting read. After clustering these, we have randomly selected 1,000 pages and found similar pages for them using our three clustering algorithms and the search-based approach.

In addition, we have randomly selected 250 (other) pages for testing the keyword extraction. Again, we have found similar pages for these, this time using Apriori[3] with either keywords generated by Synergy or by pure truncation.

Before performing the actual user test, we have removed ambiguous articles that have more than one main topic from both sets, to prevent these from confusing the users and thus polluting the user test.

11.1.2 Test Design

To reach the widest set of participants, we have used an electronic web-based[4] user test that presents the user with a page and two or four sets of similar page lists, each containing (in ranked order) 10 similar pages found by a given algorithm. For the tests of the three clustering algorithms and the search-based approach, 4 sets of similar pages are presented, while only 2 sets are presented for the tests concerning the keyword extraction methods. The idea is then that the user should choose the set of similar pages that he/she feels best matches the original page.

The different algorithms are presented in random order in order to make the test as "blind" as possible. Furthermore, the pages to test are also presented in random order, and the user

[3] Since this algorithm (together with GALOIS) is the most sensitive to overlap between keywords.

[4] We would like to thank Allan Thræn and Michael Viehweg from Mondosoft for their help in setting up the back-end needed to perform the web-based test.

can continue testing until he/she chooses to end the test by choosing “stop test” or closing the window. For screenshots of a test and the initial demographic questionnaire that the user is presented with, see appendix M.

The initial demographic questionnaire classifies the users using the following criteria:

- Age
- Gender
- Experience level with information retrieval:
  - **Novice:** Users that have little or no experience using search engines.
  - **Intermediate:** Users that are using Google or equivalent (e.g. Yahoo) on a regular basis.
  - **Expert:** Users that have experience with multiple search engines, and know how to perform advanced searches.
  - **Guru:** Users that have participated in creating a search engine.

Test participants were invited using the email shown in appendix L that briefly explains the objective of the test, and gives a small guide to participating in the test. The email was sent to employees of Mondosoft, faculty of IMM[5], friends and family. The recipients were invited to participate in the test whenever it was convenient within a three-week period.

11.1.3 Test Evaluation

To statistically evaluate the results of the user test, we have used a binomial model[6]. Using this model, we will be able to determine which statistically significant conclusions that can be drawn from the participants’ choices.

11.2 Optimisation of Algorithm Parameters

Since the clustering algorithms as well as the preprocessing and the keyword extraction stages all depend on several parameters (settings), we briefly motivate our choice of these below.

We have chosen to use a minimum document frequency of 10 for terms and bigrams, since our sensitivity analysis shows that the number of terms decreases by an order of magnitude using this setting compared with no minimum frequency, thus significantly speeding up our algorithms. Choosing a higher minimum frequency does not make sense, since we need rare terms for the small clusters that our similar page method requires. Finally, most of the words with a low document frequency are either spelling mistakes, or words unsuited for clustering, since too few other documents contain them.

[5] Informatics and Mathematical Modelling, the institute at the Technical University of Denmark that we are associated with.

[6] We would like to thank our supervisor, Jan Larsen, for suggesting and providing this model.

The maximum document frequency is set to 50%, since words having a document frequency this high must be considered stop-words, especially for a collection like Wikipedia. We have also chosen to filter out 1-character words and numbers with less than 4 digits, since these do not make meaningful keywords.

Since research indicates that bigram terms improve clustering, we have enabled bigram extraction based on scheme 2, since we do not have behaviour tracking data (needed for scheme 1) available for Wikipedia.

For weighting, we have used Mondosoft's weighting as local weight function, since our informal tests show that this more advanced scheme captures better keywords for the documents. For global weighting, we have used log-entropy, since there seems to be some indication that LSA works especially well with this method. The pseudo-weight assigned to bigram terms[7] is optimised so that the percentage of bigrams assigned as keywords is the same as when using pure term frequencies.

For Synergy, we have used an LSA dimension of 300, since this value is a good compromise as discussed earlier (in section 3.2). We also normalise the document vectors to compensate for the difference in length of Wikipedia articles before running Synergy. We have chosen to extract a minimum of 10 keywords per page, to ensure sufficient overlap between pages to form the needed small clusters. To reduce noise and to make computations efficient (especially for Apriori), we have set a maximum number of 20 keywords per document. Finally, we have used a keyword cutoff ratio of 0.5, since preliminary testing has shown that this produces good cutoffs as discussed in section 6.1.6.

For K-Means and PDDP, the maximum number of partitionings is set to 90% of the document collection size (i.e. in this case 9,000) to ensure that the desired small clusters are generated. For Apriori, we have likewise set the minimum support to 5 to ensure small clusters. For both PDDP and K-Means, Cosine is chosen as distance measure, due to the advantages of this measure mentioned in section 7.1.6.

The above determined settings are listed in appendix F.

11.3 Results: Demographics

Given the relatively short time span of the user test, we were unfortunately not able to collect enough data/samples for analysing the differences between the demographic groups. However, the demographic data gives an impression of the kinds of test participants we had. All in all, 33 persons participated in the test, performing 158 cluster algorithm comparisons and 38 extraction algorithm comparisons.

As can be see in the chart in figure 11.1, we have an over-representation of men in the test. With regard to experience, the distribution (seen in figure 11.2) matches our expectations: Few novices and gurus compared to more intermediates and experts. Finally, we also had an over-representation of "young" people as can be seen in figure 11.3.

Since most users only evaluated a rather limited number of pages, it is difficult to determ-

[7] To compensate for the fact that MondoSearch™ does not provide a weight for bigrams.

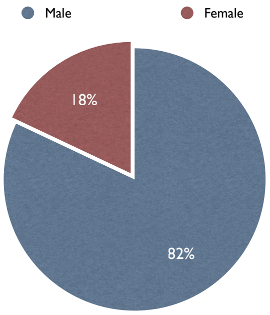


Figure 11.1: *The gender distribution.*

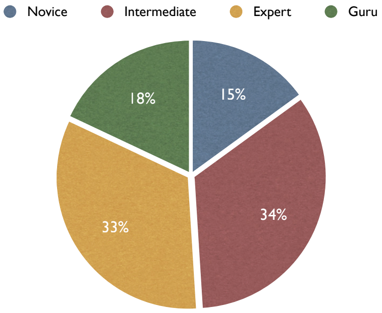


Figure 11.2: *The experience distribution.*

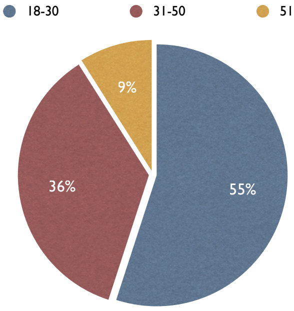


Figure 11.3: *The age distribution.*

ine whether the results contain any learning effects. For the few longs tests the participants performed, we have not been able to identify any significant difference between the choice of algorithms in the beginning and the choice of algorithms in the end.

11.4 Results: Finding Similar Pages

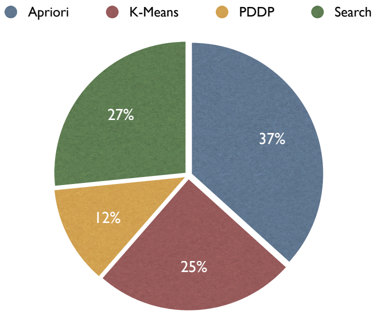


Figure 11.4: *The results of the algorithm user test.*

The results of the test concerning our three clustering algorithms and the search based approach are shown in figure 11.4.

Applying the binomial model on the results, we can conclude the following:

- On a 4.4% significance level, there is evidence that the users preferred Apriori to the search-based approach.
- On a 2.1% significance level, there is evidence that the users preferred Apriori to K-Means.
- On a 0.1% significance level, there is strong evidence that the users preferred the search-based approach to PDDP.
- On a 0.3% significance level, there is strong evidence that the users preferred K-Means to PDDP.
- We cannot reject (the null-hypothesis) that K-Means and the search-based approach find similar pages of equal quality.

The implemented clustering algorithms' ability to find similar pages can thus be ranked in the following order:

1. Apriori
2. K-Means
3. PDDP

For the small clusters used in connection with search result clustering, Apriori hence seems to perform the best. However, before discarding K-Means and PDDP, tests should also be made to test how well the algorithms perform for larger clusters (e.g. in connection with search result clustering).

Also, two of our algorithms are as good as or better than the search-based approach, which is a strong indicator that using clustering for finding similar pages is a feasible and promising solution. Especially, if the clustering is already available because it is used for other purposes (e.g. result clustering). On the other hand, the search-based approach did not perform that bad either and since it is simpler to implement it should probably not be discarded yet. Research should be undertaken to determine whether the search-based approach could be improved by using better search-terms, selected with another algorithm than pure truncation based on local weights[8].

11.5 Results: Comparing Synergy and Pure Truncation

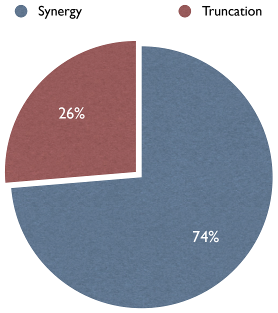


Figure 11.5: *The results of the keyword extraction user test.*

The results of the test concerning Synergy and pure truncation are shown in figure 11.5. The results clearly indicate that the more advanced LSA-based approach (including filtering, bigrams and global weighting) also for small clusters yields better results than pure truncation based on MondoSearch's local weighting.

If we apply the binomial model on the results, we find strong statistical evidence that users prefer Apriori using Synergy instead of pure truncation on a 0.1% significance level.

[8] It could be interesting to see how "Synergy plus search" would perform.

Chapter 12

Conclusion and Future Work

12.1 Conclusion

The overall aim of this project was to help Mondosoft to evaluate the quality and feasibility of using clustering to improve their information retrieval products. More specifically focusing on applying offline clustering to find similar pages and, to a lesser extent, for clustering search results. In order to achieve this objective, we have designed and implemented a clustering toolkit based on the results of a comprehensive literature study and analysis. The analysis has focused on research into both clustering and the information retrieval process that clustering is a part of. Our main contributions thus consist of the implemented clustering toolkit – including several novel ideas/approaches of our own – and the findings we have obtained using this toolkit.

Implemented Toolkit

Based on the above analysis, we chose to continue working with 5 promising clustering algorithms spanning different classes of clustering methods. We further decided to use an unconventional approach, spliting the clustering stage into two stages: Keyword extraction and keyword-based clustering. This was done in order to filter out noise, to allow lattice based clustering and to make clustering fast and thus feasible in connection with large websites.

We have designed the clustering toolkit to be both flexible and modular, allowing experiments combining different algorithms and easy addition of additional (clustering) algorithms later. We have also fulfilled the requirements of integration with Mondosoft's products as specified in the problem definition (see section 2.3.2).

The clustering toolkit incorporates the following major components covering the entire clustering process from data extraction to postprocessing:

- Data extraction directly integrated with MondoSearch™.
- A preprocessing module including filtering, stemming using tries, two novel methods for bigram extraction (one of these utilising data from BehaviorTracking™) and various kinds of weighting functions.

- A keyword extraction module using our own novel approach based on Latent Semantic Analysis.
- Modules implementing prototypes of the 5 promising clustering algorithms, namely: K-Means, CURE, PDDP, GALOIS and a novel Apriori-based fast method for building Galois lattices.
- A postprocessing module including functions for generating similar pages and a preliminary approach for generating search result clustering.

Main Findings

Using the implemented toolkit, we have made an assessment of the performance of the different algorithms with regard to running-time and memory consumption. We have found that the implemented preprocessing module scales roughly linear (in the number of pages) both with regard to running-time and memory.

We have also found evidence that the number of terms added after filtering seems to scale degressively with the size of the document collection, even in wide-spanning collections like Wikipedia. The average number of new terms per added document seems to stabilise at less than 1. Our experiments also show that for large collections, stemming based on the implemented Porter stemmer reduced the term space by more than 20%. It is unknown whether this percentage will decrease for larger collections. This is unfortunately likely, since new terms added for large collections will increasingly be proper nouns, spelling mistakes etc. that neither can nor should be stemmed.

The implemented LSA-based keyword extraction approach (Synergy) consists of two computationally expensive processes: The Singular value decomposition of the term-document matrix, and the extraction of keywords by constructing an approximated term-document matrix from the leading singular values and vectors. For the SVD, we have found that the running-time scales more or less linearly with the number of non-zero entries (i.e. the number of term-document relations) in full accordance with the theory. We have determined that the extraction part scales in very strict accordance with the theoretical complexity. It scales with the number of documents times the number of terms. This part of the extraction should benefit from the decreasing number of new terms per document as the collection size grows beyond the size that we have measured. For both parts of the Synergy keyword extraction scheme, the memory consumption scales near-linearly.

The empirical evidence found during our testing strongly suggests that both CURE and GALOIS are unfeasible in terms of running-time. Both algorithms scale quadratically and involve very large constant factors. K-Means and PDDP also ran in quadratic time (as expected), but with low constants. The running-time of these was quadratic, since we desire clusters of a fixed minimum size and not a fixed number of clusters. Both K-Means and PDDP had excellent memory performance scaling roughly linearly with the number of documents. The running-time of Apriori is (as expected) remarkably good – it scales near-linearly with low constants when using a fixed minimum support threshold. Apriori should run even faster if the minimum support was to be given as a percentage of the collection size. However, Apriori has a relatively high memory consumption, which is partly due to minor shortcomings in the current implementation.

For the lattice-based algorithms, we have found that for Wikipedia (which spans a very large knowledge domain) the lattice size seems to grow linearly with the size of the document collection. On the other hand, for a single-domain collection spanning information science abstracts (CISI), we found that the number of lattice nodes added per new document stabilises much earlier. For low LSA-dimensions (used during keyword extraction), we even found evidence of saturation effects[1] when clustering CISI.

We have also found evidence that the lattice size depends greatly on the LSA dimension used in the keyword extraction stage. A high LSA-dimension yields small lattice sizes due to low overlap between the keywords assigned to the individual documents. Surprisingly, a low dimension also yields a small lattice, this time due to saturation effects. The maximum lattice is created exactly when maximum keyword overlap is achieved and saturation effects avoided. Empirically we found this to be at a relatively low dimension for the CISI collection. Finally, we have also found that the lattice size depends on the average number of keywords assigned per document - it scales quadratically or worse with the average number of keywords. The lattice-based algorithms thus depend on a limited number of keywords to be feasible.

We have shown that our approach to finding similar pages scales linearly for a single page and thus quadratically for the whole collection when constructing a table of similar pages.

During our work on implementing and experimenting with the clustering toolkit, we have discovered that clustering webpages (contrary to clustering pure text documents) presents some challenges due to the way typical webpages are made. These challenges mean that we have had to make adjustments to the way we filter and extract data, specific to each of the sites that we have tested, to gain maximum cluster quality. Because of this, we believe that it would be difficult to make a product based on clustering that works optimally “out-of-the-box”, without having to first fine-tune it to the given site.

The performed sensitivity analysis shows that the extracted keywords and thus the subsequent clustering is quite sensitive to most preprocessing parameters. With regard to the choice of LSA-dimension, we see the expected effect that the lower the chosen dimension is, the more keywords are inferred (i.e. keywords not in the document that they are assigned to), and the lower the size of the space spanned by the assigned keywords becomes. Furthermore, our tests indicate that a minimum document frequency threshold of 10 reduces the number of terms by an order of magnitude, and thus improves the running-time of the keyword extractor significantly.

To test the capabilities of the implemented algorithms[2] with regard to finding similar pages, we have conducted a web-based user test. This user test was to establish how our algorithms performed compared to each other and how they performed compared to a simpler search-based approach. In addition, the test should uncover how the elaborate LSA-based keyword extraction approach performed in connection with subsequent clustering, compared with simple truncation of words based on local weights. 33 persons participated in the user test, making a total of 196 evaluations of similar pages. Fortunately, this provided an adequate amount of data for making statistically valid conclusions about the above.

On a statistically significant level, we have established that the tested algorithms rank in the following order: Apriori, K-Means and PDDP. This indicates that the lattice-based approach of Apriori seems to outperform (in terms of quality) the partition-based approaches used by K-Means and PDDP. However, further studies should be conducted to establish whether Apriori

[1] I.e. the average number of nodes added per documents starts decreasing.

[2] The 3 feasible algorithms: K-Means, PDDP and CURE.

is also better for larger clusters (for instance in connection with search result clustering).

We also found statistically significant evidence that Apriori performs better than the search-based approach. However, based on the statistical analysis, we were not able to determine whether the search-based approach or K-Means produce the best similar pages. Thus, two of our algorithms currently seem to be as good as or better than the search-based approach, which indicates that using clustering for finding similar pages is a both feasible and promising approach. However, the search-based approach should not be discarded either, since it is much simpler to implement and might be improved using another scheme for selecting the search terms.

With regard to Synergy, the LSA-based keyword extraction scheme, we have found strong evidence that this scheme (when used for subsequent clustering) produces small clusters of far higher quality[3] than the simple truncation of terms based on local weight.

Finally, we have made a small test/demonstration of our preliminary search result clustering approach. The results of this look very promising with high precision and recall for the relevant clusters within the search results. However, further studies should be conducted to determine whether this (for words that are not as ambiguous as our samples) lead to better search effectiveness, efficiency and experience.

All in all, we consider our implemented clustering toolkit and the findings we have made using this a considerable success. A success that Mondosoft should be able to use as a foundation for further work and studies within the field of clustering. Hopefully this will lead to actual products, with notable benefits for their users and customers, down the road.

12.2 Recommendations to Mondosoft

Since we now approach the end of this project, it is time to "pass on the baton" to Mondosoft. This section is therefore meant to assist Mondosoft in converting our work and research into actual products and benefit for their customers.

12.2.1 Finding Similar Pages

With regard to finding similar pages, we recommend the following:

1. First, research should be carried out to assess the impact of using a more-like-this feature in connection with Mondosoft's existing search engine. How can it be used to best improve the search efficiency, effectiveness and experience for the users (if at all)?

2. If it turns out that the above feature yields notable improvements, a decision should be made whether to use clustering for this or the less complex search-based approach. Our research shows that Apriori-based clustering is significantly superior to the simpler approach. However, attempts might still be made to try to improve the quality of the search-based approach using better search terms, since this approach is much simpler than actual clustering.

[3] At least when the clustering is evaluated based on the similar pages that it is able to find for a given document.

3. Finally, a prototype product should be implemented, and, if the decision to use clustering is made, an optimised version of the Apriori clustering scheme should be utilised.

12.2.2 Search Result Clustering and Beyond

With regard to search result clustering, our preliminary experiments show that this is a very promising approach for improving recall and precision at least in connection with ambiguous searches. We recommend that the following steps are taken:

1. First, a more finished prototype based on our preliminary approach should be designed and implemented (working with the three feasible clustering algorithms).
2. This prototype should be integrated with MondoSearch™, and studies should be made to determine if search result clustering generally improves the search efficiency, effectiveness and experience. It should further be determined, both from a technical and a user-minded standpoint, how this is best done.
3. The offline clustering-based approach discussed in this report should, if possible, also be compared with online methods for directly creating search result clustering. This will help establishing whether the online alternative (e.g. suffix-tree clustering), which is a more direct approach, might yield comparable results.

In addition to search result clustering, some of the other promising areas for application of clustering (see section 2.2.2) should also be considered in connection with improving and expanding Mondosoft's suite of products:

- Investigate how clustering can be used as an automatic alternative to manually building, maintaining and updating a Yahoo-like information hierarchy.
- Investigate how our implemented lattice clustering can be used in connection with guided/interactive searches, allowing users to browse through the information content of a site by adding or removing recommended search terms (keywords) on-the-fly. E.g. an improved version of what Gigablast[4] currently is doing.
- Investigate if the keywords found using Synergy can be used for other applications than clustering. E.g. auto-tagging documents with keywords.
- Investigate whether it is feasible to automatically classify documents based on a given taxonomy utilising keywords that we have assigned using Synergy and some ideas from clustering.

12.2.3 Recommended Enhancements

Due to the inherent limited time frame of this project, we have mainly focused on building experimental prototypes and the current implementations should hence be enhanced and optimised before they can form the basis of actual products.

[4] `http://www.gigablast.com/`

Besides from improving data structures and optimising algorithms in general (especially in terms of the memory consumption of Apriori), the stemming algorithm should be made multi-lingual, since Mondosoft targets several languages. We hence recommend that a multi-language stemmer is implemented that integrates with our implemented clustering. This could either be realised using Porter-like stemmers designed for the target languages, or, using another, more language-independent, approach. All other parts of our implemented modules are multi-lingual by design, and should thus be ready to be integrated in multi-lingual products.

Another practical enhancement that we recommend for implemenation is a scheme to control the keywords assigned to documents, where a "white-list" can be used to "boost" the weights of some keywords and a "gray-list" to decrease the importance of others. In practice, this could be a number between -1 and 1, assigned to chosen words, where -1 means that the word is never assigned as keyword and 1 means that it is always assigned as keyword if it is present on the page. Values between -1 and 0 should be used to relatively decrease the weight of the given word, whereas values between 0 and 1 should be used to relatively boost the weight of the given word.

12.3 Future Work

As well as passing on the "practical" baton to Mondosoft, the academic implications of our thesis work lead to several interesting opportunities for future work and research. Below, we outline some of these that naturally follow from our work and the conclusions that we have made:

- Further research should be done into how different keyword extraction schemes work with the various clustering algorithms, in order to establish whether our LSA-based approach is the best suited for clustering.

- Further studies should be made, investigating how our two-stage approach compares in quality to clustering algorithms using the full term space (e.g. K-Means using full term space vs. K-Means using keywords).

- Alternative and more automatic ways of evaluating the clusters generated by our algorithms should be investigated as a complement to more user-focused methods like user-tests, since this can be used as guidelines for the development. However, as already touched upon, we strongly believe that the end result should be evaluated according to the experiences, opinions and behaviour of actual users.

- A study comparing manually assigned keywords with the ones assigned by Synergy when used in connection with lattice-based clustering[5] (Apriori).

- The methods investigated in this report primarily consider documents as bags of words (and bigrams). They should thus be compared with methods that utilise sentence structure and word order to a greater extent (e.g. suffix-trees and approaches using linguistic knowledge).

[5] Manually assigned keywords cannot be directly used with clustering algorithms utilising the vector model, since this model requires that each keyword-document relation is assigned a weight.

- Further research should be carried out to establish the impact of using bigrams (and n-grams in general) to see if bigrams are an advantage to clustering applications, and whether the added benefits outweigh the disadvantage of a larger term space.
- Experiments should also be carried out using a lattice based approach (e.g. Apriori) to cluster other features than keywords. This could for instance be queries that have led to the user finding the document (and choosing it).

In addition, research into new methods of clustering that combine speed, low memory consumption and quality in a feasible mix is still a relevant and interesting challenge.

12.4 Future Perspectives

Having worked intensively with information retrieval in general and clustering in particular, it is our belief that clustering definitely has an important role to play in the future of information retrieval. Clustering represents a way of discovering structure in the seemingly unstructured data that comprises the vast web of information known as the Internet.

Since modern search engines, like Google and MSN, are operating near the current limits for information retrieval in unstructured data, we believe that the next generation of information retrieval tools are going to be increasingly based on automatically and manually detecting structure in online information. Such structure could take the form of categories (cf. Yahoo), clusters (which, in a sense, are automatically detected categories) and other machine-readable metadata (i.e. the much talked about Semantic Web).

The problems of low precision and "information overload" will continue to increase as the amount of information available in electronic form on the Internet increases. To face up to this challenge, new ways of increasing precision in connection with information retrieval should be sought, and in this respect, clustering seems like a promising research area. Clustering offers an automatic alternative to building, maintaining and updating an information hierarchy for a single site – or the entire Internet – which can be used to improve and ease the information retrieval process.

In the future, everyone in our information-rich societies will increasingly be knowledge workers, depending on finding and retrieving the right information at the right time. If clustering can be harnessed to aid this process, it could be a part of the solution ensuring that the promises of the Internet revolution turn into real benefits and not just the nightmare envisioned by Roussinov and Chen in the quote from section 2.1:

> "Our productivity in generating information has exceeded our ability to process it, and the dream of creating an information-rich society has become a nightmare of information overload."

Bibliography

[AFMiA00] El-Sayed Atlam, Masao Fuketa, Kazuhiro Morita, and Jun ichi Aoe. Similarity measurement using term negative weight and its application to word similarity. *Inf. Process. Manage.*, 36(5):717–736, 2000.

[AS94] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo, editors, *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, pages 487–499. Morgan Kaufmann, 12–15 1994. Available at `http://citeseer.ist.psu.edu/agrawal94fast.html`.

[Bak04] Zachary Baker. Efficient algorithms and architectures for high-speed text processing and data correlation. Available at `http://halcyon.usc.edu/~zbaker/quals.pdf`, 2004.

[BdMAS02] Vicente P. Guerrero Bote, Felix de Moya Anegon, and Victor Herrero Solana. Document organization using kohonen's algorithm. *Inf. Process. Manage.*, 38(1):79–89, 2002.

[BEX02] F. Beil, M. Ester, and X. Xu. Frequent term-based text clustering. In Proc. 8th Int. Conf. on Knowledge Discovery and Data Mining (KDD), 2002. Available at `http://citeseer.ist.psu.edu/beil02frequent.html`.

[BG04] A. Banerjee and J. Ghosh. Frequency-sensitive competitive learning for scalable balanced clustering on high-dimensional hyperspheres. *Neural Networks, IEEE Transactions on*, 15(3):702–719, 2004.

[BN04] Christian Borgelt and Andreas Nuernberger. Experiments in document clustering using cluster specific term weights. In *Proc. Workshop Machine Learning and Interaction for Text-based Information Retrieval (TIR 2004, Ulm, Germany)*, pages 55–68, 2004.

[Bod03] Ferenc Bodon. A fast apriori implementation. In Bart Goethals and Mohammed J. Zaki, editors, *Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations (FIMI'03)*, volume 90 of *CEUR Workshop Proceedings*, Melbourne, Florida, USA, 19. November 2003. Available at `citeseer.ist.psu.edu/bodon03fast.html`.

[Bol98] Daniel Boley. Principal direction divisive partitioning. *Data Mining and Knowledge Discovery*, 2(4):325–344, 1998.

[Bra02] Matthew Brand. Incremental singular value decomposition of uncertain data with missing values. In *European Conference on Computer Vision (ECCV)*, volume 2350, pages 707–720, May 2002. Available at `http://www.merl.com/reports/docs/TR2002-24.pdf`.

[Bra03] Matthew Brand. Fast online svd revisions for lightweight recommender systems. In *SIAM International Conference on Data Mining (SDM)*, May 2003. Available at `http://www.merl.com/publications/TR2003-014/`.

[BYRN99] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley/ACM Press, 1999.

[BYSZ02] Wu Bin, Zheng Yi, Liu Shaohui, and Shi Zhongzhi. Csim: A document clustering algorithm based on swarm intelligence. In *Proceedings of the 2002 Congress onEvolutionary Computation (CEC '02)*, 2002.

[CLRS01] Thomas H. Cormen, Charles L. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, second edition, 2001.

[CR95] Claudio Carpineto and Giovanni Romano. Automatic construction of navigable concept networks characterizing text databases. In *Proceedings of the Fourth Congress of the Italian Association for Artificial Intelligence*, pages 67–78, October 1995.

[CR96] Claudio Carpineto and Giovanni Romano. A lattice conceptual clustering system and its application to browsing retrieval. *Machine Learning*, Volume 24, Issue 2:95 – 122, May 1996. Available at `http://search.fub.it/claudio/pdf/ML1996.pdf`.

[CSB03] Guihong Cao, Dawei Song, and Peter Bruza. Suffix tree clustering on post-retrieval documents, July 2003. Available at `http://www.dstc.edu.au/Research/Projects/Infoeco/publications/tech-report-suffix-tree.pdf`.

[CTN03] Liang Chen, Naoyuki Tokuda, and Akira Nagai. A new differential lsi space-based probabilistic document classifier. *Information Processing Letters*, Vol.88 Issue.5:203–212, August 2003. Available at `http://azalea.sunflare.co.jp/rd_dir/papers/2003-IPL-classification.pdf`.

[DM02] Mathieu Delichere and Daniel Memmi. Neural dimensionality reduction for document processing. In *ESANN'2002 proceedings - 10th European Symposium on Artificial Neural Networks.*, pages 211–16, 2002. Available at `http://www.dice.ucl.ac.be/Proceedings/esann/esannpdf/es2002-252.pdf`.

[dSdSBJS$^+$04] Paulo S. L. de Souza, Alceu de S. Britto Jr, Robert Sabourin, Simone R. S. de Souza, and Dibio L. Borges. K-means vq algorithm using a low-cost parallel cluster computing, 2004. Available at `http://www.livia.etsmtl.ca/publications/2004/Souza_PDCN_2004.pdf`.

[Dum93] S. Dumais. Lsi meets trec: A status report. In *The First Text REtrieval Conference (TREC1), National Institute of Standards and Technology Special Publication 500-207*, pages 137–152, 1993. Available at `http://lsi.research.telcordia.com/lsi/papers/TREC1.ps`.

[FBY92] William B. Frakes and Ricardo Baeza-Yates. *Information Retrieval : Datastructures and Algorithms.* Prentice-Hall, 1992.

[GEG03] Daniel Jimenez Gonzalez, Carlos F. Enguix, and Vicente Vidal Gimeno. A comparison of experiments with the bisecting-spherical k-means clustering and svd algorithms, 2003. Available at `http://www.fiv.upv.es/jotri/Ponencias/Comparison.pdf`.

[GLY03] Deyun Gao, Xiaohu Li, and Zheyuan Yu. Parallel clustering of large document collections, July 2003. Available at `http://flame.cs.dal.ca/~zyu/courses/6702/LiteratureReview.pdf`.

[Gro] Leif Gronqvist. Latent semantic indexing and beyond. Available at `http://www.gslt.hum.gu.se/~leifg/doc/nodalida_gronqvist.html`.

[GRS98] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. Cure: an efficient clustering algorithm for large databases. In *ACM SIGMOD International Conference on Management of Data*, pages 73–84, June 1998.

[HSS03] A. Hotho, S. Staab, and G. Stumme. Explaining text clustering results using semantic structures. In *7th European Conference on Principles of Data Mining and Knowledge Discovery (PKDD 2003)*, 2003. Available at `http://www.kde.cs.uni-kassel.de/hotho/pub/HothoStaabStumme_ExplainingTextClustering.pdf`.

[Hul03] Anette Hulth. Improved automatic keyword extraction given more linguistic knowledge. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP'03)*, July 2003. Available at `http://www.dsv.su.se/~hulth/hulth_emnlp03.pdf`.

[HZDS01] X. He, H. Zha, C. Ding, and H. Simon. Web document clustering using hyperlink structures. Technical report, Department of Computer Science and Engineering, Pennsylvania State University, 2001. Available at `http://citeseer.ist.psu.edu/he01web.html`.

[KB02] L. Kovics and P. Baranyi. Document clustering based on concept lattice. *Systems, Man and Cybernetics, 2002 IEEE International Conference on*, Vol. 7, 2002.

[KL99] Irwin King and Tak-Kan Lau. Non-hierarchical clustering with rival penalized competitive learning for information retrieval. In *MLDM '99: Proceedings of the First International Workshop on Machine Learning and Data Mining in Pattern Recognition*, pages 116–130, London, UK, 1999. Springer-Verlag. Available at `http://www.cse.cuhk.edu.hk/~king/PUB/mldm99_2.pdf`.

[LD97] Thomas K. Landauer and Susan T. Dumais. The solution to platos problem: The latent semantic analysis theory of acquisition, induction and representation of knowledge. *Psychol. Rev.*, vol. 104:pp. 211–240, 1997. Available at `http://lsi.argreenhouse.com/lsi/papers/PSYCHREV96.html`.

[Ler99] Kristina Lerman. Document clustering in reduced dimension vector space, Januar 1999. Available at `http://www.isi.edu/~lerman/papers/Lerman99_abstract.html`.

[LFL98] Thomas K. Landauer, Peter W. Foltz, and Darrell Laham. An introduction to latent semantic analysis. *Discourse Processes 25*, pages 259–284, 1998. Available at http://lsa.colorado.edu/papers/dp1.LSAintro.pdf.

[Mar04] Moses Claus Martiny. Optimizing matrix multiplication, November 2004. Available at http://www.martiny.dk/papers/dmm.pdf.

[Miy03] Sadaaki Miyamoto. Information clustering based on fuzzy multisets. *Information Processing and Management: an International Journal*, 39(2):195–213, 2003.

[Mon02] Mondosoft. White paper on mondosearch ranking. Technical report, Mondosoft, May 2002.

[NO02] Joergen Fischer Nilsson and Nikolaj Oldager. Introduction to orders and lattices. Part of Course 02284 Knowledge-based Systems, 2002. Available at http://www2.imm.dtu.dk/~sno/kbs2.ps.

[NPM01] P. Nakov, A. Popova, and P. Mateev. Weight functions impact on lsa performance. In *Proceedings of the EuroConference Recent Advances in Natural Language Processing*, pages 187–193, September 5-7 2001. Available at http://www.cs.berkeley.edu/~nakov/selected_papers_list/nakov_RANLP01.pdf.

[PFTV92] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes in C : The Art of Scientific Computing*. Cambridge University Press, 1992.

[QSW02] Yun-Tao Qian, Qing-Song Shi, and Qi Wang. Cure-ns: A hierarchical clustering algorithm with new shrinking scheme. *Machine Learning and Cybernetics, 2002. Proceedings. 2002 International Conference on*, Vol.2:895–899, November 2002.

[RC01] Dmitri G. Roussinov and Hsinchun Chen. Information navigation on the web by clustering and summarizing query results. *Information Processing and Management: an International Journal*, Volume 37 , Issue 6 (November 2001):789 – 816, 2001.

[RG00] S. M. Rueger and S. E. Gauch. Feature reduction for document clustering and classification. Technical report, Computing Department, Imperial College, London, UK, 2000. Available at http://citeseer.ist.psu.edu/uger00feature.html.

[SS97] H. Schuetze and H. Silverstein. Projections for efficient document clustering. In *Proceedings of SIGIR'97*, pages 74–81, July 1997. Available at http://citeseer.ist.psu.edu/76529.html.

[TKUE99] Hideki Tanaka, Tadashi Kumano, Noriyoshi Uratani, and Terumasa Ehara. An efficient document clustering algorithm and its application to a document browser. *Information Processing and Management: an International Journal*, Vol.35 Issue.4:541–557, 1999.

[TVR02] Anastasios Tombros, Robert Villa, and C. J. Van Rijsbergen. The effectiveness of query-specific hierarchic clustering in information retrieval. *Information Processing and Management: an International Journal*, Volume 38, Issue 4 (July 2002):559 – 582, 2002. Available at http://www.dcs.qmul.ac.uk/~tassos/publications/ipm_02.pdf.

[YCCP] Clara Yu, John Cuadrado, Maciej Ceglowski, and J. Scott Payne. Patterns in unstructured data : Discovery, aggregation, and visualization. A Presentation to the Andrew W. Mellon Foundation. Available at `http://javelina.cet.middlebury.edu/lsa/out/cover_page.htm`.

[ZHTY03] Xiao-Shen Zheng, Pi-Lian He, Mei Tian, and Fu-Yong Yuan. Algorithm of documents clustering based on minimum spanning tree. In *Proceedings of the second International Conference on Machine Learning and Cybernetics*, volume Vol. 1, pages 199–203, November 2003.

Glossary

Affix An additional element placed at the beginning or end of a root, stem, or word, or in the body of a word, to modify its meaning. [Oxford Dictionary]

Agglomerative clustering Agglomerative clustering methods start with an initial clustering of the term space, where all documents are considered representing a separate cluster. The closest clusters (using a given inter-cluster similarity measure) are then merged continuously until only 1 cluster or a predefined number of clusters remain.

Apriori Apriori is a well-known data mining algorithm for discovering frequent itemsets in a transaction database. We have implemented an extended version of this algorithm to do lattice-based clustering.

BehaviorTracking(TM) Is an analytical reporting tool created by Mondosoft providing information on the search activity and visitor behaviour on a site using MondoSearch(TM).

Binary search In computer science, binary search is a fast search algorithm for searching a set of sorted data for a particular value. [Wikipedia]

Bisecting K-Means The divisive version of K-Means we have chosen to implement.

Boolean algebra In formal logic, mathematics and computer science, Boolean algebras are algebraic structures which "capture the essence" of the logical operations AND, OR and NOT as well as the corresponding set-theoretic operations intersection, union and complement. [Wikipedia]

Boolean model The Boolean model for information retrieval is a simple retrieval model based on set theory and Boolean algebra. In its essence, the boolean representation of a document is a set of terms, where the terms are words from the document. The term-document relations are thus of binary nature.

Chernoff bound In probability theory, the Chernoff bound gives a lower bound for the success of majority agreement for n independent, equally likely events. Chernoff bounds are used in connection with the sample-based version of CURE. [Wikipedia]

Cluster centroid The centroid of a cluster is the same as the vector-sum of all vectors in the cluster divided by the number of vectors in the cluster – i.e. "the average document of the cluster".

Cluster hypothesis The cluster hypothesis states that relevant documents, to a given information need, tend to be more similar to each other than to non-relevant documents, and therefore tend to appear in the same clusters.

Collection frequency *See* Global frequency.

Complete lattice In mathematics, a complete lattice is a partially ordered set in which all subsets have both a supremum (join) and an infimum (meet). [Wikipedia]

Conflation The process of combining syntactic variations of words is known as conflation and is a more general term for stemming.

Connecting result The connecting result states that any lattice can be conceived of as either a partial order or as an algebra, or as both.

Core-terms Terms extracted from a document that capture the essence of the document's main topic/theme. Often we use the term "keywords" since we still think that this is the notion that comes closest to describing the terms we want to extract.

Cosine distance measure This distance measure is often used in clustering and measures the angular separation of documents.

Crawling Is the process where the links of a given set of websites are traversed to gather all relevant pages for later indexing.

Crawl-time filtering Crawl-time filtering includes all filtering rules that can be set up before actually crawling and indexing the data. This includes stop-lists, numbers not falling into specific formats etc..

CURE CURE, Clustering Using REpresentatives, is an agglomerative hierarchical clustering method that uses a single-link distance measure between clusters. It is one of the clustering algorithms that we have implemented.

Distance measure A distance measure (or, dually, similarity measure) lies at the heart of document clustering. Several ways for measuring the distance between two documents exist, some are based on the vector model (e.g. Cosine distance or Euclidean distance) while others are based on the Boolean model (e.g. size of intersection between document term sets).

Divisive methods Divisive clustering methods start with a single cluster containing all documents. These methods continuously divide clusters until all documents are contained in their own cluster or a predefined number of clusters are found.

Document classification Assigning documents into predefined groups/categories.

Document clustering We define clustering as the automatic discovery of document clusters/groups in a document collection, where the formed clusters have a high degree of association between members, whereas members from different clusters have a low degree of association.

Document frequency The number documents in which a given term occurs.

Entropy The basic concept of entropy in information theory has to do with how much randomness there is in a signal or random event. An alternative way to look at this is to talk about how much information is carried by the signal. [Wikipedia]

Euclidean distance measure This measure is based on the geometrical distance between two documents in a Euclidean multidimensional space.

Extent The extent of a formal concept describes the subset of objects that belong to the concept. A cluster's extent consists of precisely those documents that contain the cluster's intent as a subset of their assigned keywords.

Features Are a generic notion used to describe terms or other information extracted from a document with the intent of using it for clustering.

Filtering In clustering filtering of irrelevant words can reduce the size of the database as well as serving the purpose of removing noise from the database.

Finding similar documents Finding similar documents is a popular way of enhancing the search experience, allowing the user to identify pages containing desired information easier. This is often implemented as a "similar pages" link next to each search result.

Flat clustering Clustering methods simply resulting in all the documents of the document space ending in different clusters, without any hierarchy, are called flat clustering methods.

Formal concept A formal concept is defined either by an intent or by an extent, or both. The intent is a set of keywords that defines the concept and the extent is exactly the documents that have these keywords in common.

Formal concept analysis (FCA) An emerging area that has many promising applications within data analysis, information retrieval and knowledge discovery. The central notions in formal concept analysis are formal context and formal concept.

Formal context Is a triple consisting of a set of object, a set of attributes and a binary relation between the two former sets.

Frequent itemset A frequent itemset is a set of items that often (more than a given minimum times) occur in the transactions in a given database.

Frobenius norm The Frobenius norm is a matrix norm of an $m \times n$ matrix defined as the square root of the sum of the absolute squares of its elements. [MathWorld]

GALOIS GALOIS is a conceptual clustering method that generates soft clusters arranged in a lattice structure. The algorithm solves the problem of automatically generating what is known as a Galois lattice from a given set of objects – documents in our case – each with a set of attributes – keywords in our case. GALOIS is one of the clustering algorithms we have implemented.

Galois lattice A Galois lattice represents a conceptual hierarchy of the objects in a given formal context, based on the object's attributes within that context.

Global frequency The number of times that a given term occurs in the document collection. This is sometimes also called collection frequency.

Global weight function A global weight function represents the weight of a term in the entire document collection, and is a measure of how important the given term is as a descriptor of documents in the given collection.

Hard clustering A clustering where clusters are non-intersecting. Also sometimes referred to as a partitioning.

Heap In computer science a heap is a specialized tree-based data structure. Heaps must fulfil a heap property (e.g. min-heap). Heaps are used to implement priority queues, sorting etc.

Hierarchical clustering Hierarchical clustering approaches attempt to create a hierarchical decomposition of the given document collection thus achieving a hierarchical structure.

Incremental algorithm An algorithm producing results as the input is added one entry at a time, allowing easy updating when new inputs are available.

Indexing In document clustering, indexing is responsible for building indexes of term/document relations. Such indexes form the basis for the term-document matrix that is often the starting point for the clustering process.

Infimum In mathematics the infimum of a subset of some set is the greatest element, not necessarily in the subset, that is smaller than all other elements of the subset. Consequently the term greatest lower bound is also commonly used. In lattice terminology this is found using the *meet* operation.

Information Retrieval Information Retrieval (IR) is an emerging subfield of information science concerning representation, storage, access and retrieval of information.

InformationManager(TM) Is a product from Mondosoft the utilises that data from BehaviorTracking(TM) to allow web managers to refine content and search experience when using MondoSearch(TM).

Intent Formally, the intent is a set of attributes that defines a formal concept. In lattice-based clustering the intent consists of exactly the keywords that the documents in the extent of the cluster have in common.

Interactive search A form of search where the user drills down and finds the desired information step-by-step by gradually refining the search.

KD-tree A multidimensional search tree for points in a multidimensional space. Levels of the tree are split along successive dimensions at the points.

Keywords In this thesis keywords refer to the core-terms that we extract from documents to capture the essence of their main topic/theme.

Keyword cutoff scheme Simple scheme implemented to ensure that the number of keywords assigned to a given page is variable depending on the weight distribution of discovered keyword candidates.

K-Means A very straightforward clustering algorithm that partitions the documents in the term space into K clusters, hence the name. The algorithm produces a hard, non-hierarchical clustering. K-Means is one of the clustering algorithms that we have implemented.

Lanczos method A fast numerical methods for performing reduced SVD .

Learning effects A notion describing the situation where the answers of test participants to the first questions/tests differ significantly from the ones later given – because the participants gradually "learn" how they like to approach the test questions.

Local weight function The local weight function represents the weight of a given term in a given document and is a measure of how related the term and document are.

LSA Is a statistical/mathematical method for indexing, retrieval and analysis of textual information. It is based on reduced SVD and the underlying dimensional reduction.

LSA-dimension Notion used to describe the number of leading singular triplets that is kept when performing LSA.

LSA-subspace The latent subspace spanned by the leading singular vectors. This subspace is of a lower dimension thatn the full term-document space.

Manhattan distance measure The Manhattan distance between two points in a Euclidean space with a fixed Cartesian coordinate system is defined as the sum of the lengths of the projections of the line segment between the points onto the coordinate axes.

Model-based clustering Model-based clustering is based on hypothesising models for clusters in the document collection, and then for each document finding the cluster whose model the document best fits.

MondoSearch(TM) Is Mondosoft's multi-lingual search engine targeted at search within individual (large) sites.

Mondosoft Mondosoft is a software company offering a suite of enterprise search, analytics and site optimisation products.

More-like-this *See* finding similar documents.

Neural networks Neural networks are designed to emulate the internal functionings of the human brain, more specifically, the neurons. Neural network-based methods rely on either supervised or unsupervised learning to assigning multi-dimensional outputs to multi-dimensional input.

Noise Irregular fluctuations and patterns that accompany data, but tend to obscure and not contain meaningful data or other information. [Oxford Dictionary]

Offline clustering Creating and storing clusters in a fast database once and for all, and only leaving simple operations to be performed when querying the database.

Online clustering Online clustering is performed on-the-fly when the clustering is needed.

Outlier Documents that are not particularly related to any other documents in the collection, these documents can thus be considered as noise when performing document clustering.

Overstemming Overstemming is a common problem when using stemming algorithms and occurs when two unrelated words such as "engineer" and "engine" are conflated.

Partial order In mathematics, a partially ordered set is a set equipped with a binary partial order relation, formalising the intuitive concept of a (not necessarily total) ordering. [Wikipedia]

Partition-based clustering Partition-based clustering algorithms partition a document collection into a number of hard clusters using a given distance measure.

PDDP PDDP, Principal Direction Divisive Partitioning, is a fast hierarchical clustering algorithm that we have implemented.

Polysemy The coexistence of many possible meanings for a word or phrase. [Oxford Dictionary]

Post-crawl filtering Is filtering that is done after the crawling using the global to filter out unwanted terms.

Postprocessing The actual applications of a document clustering to some purpose within information retrieval is what we refer to as "postprocessing".

Precision Refers to the common information retrieval metric defined as the number of relevant documents in the result compared to the total number of documents in the result.

Prefix One or more characters in the beginning of a word.

Preprocessing Various subprocesses concerned with adapting the indexed document collection for clustering purposes are what we refer to as “preprocessing techniques”.

Pruned Galois lattice The Galois latice that is produced by our extended version of Apriori. This lattice only contains the concepts that have a specified minimum extent size.

Quicksort Fast sorting algorithm that is often used.

Recall The common metric used in information retrieval. It is defined as the number of relevant documents found compared with the total number of relevant documents.

Saturation effect The situation where the average number of lattice nodes added per new document starts to decrease as a result of saturation of the conceptual domain spanned by the document collection.

Search result clustering Using Document clustering to automatically categorise a search result into topic groups (clusters).

Similarity measure *Dual to* distance measure.

Soft clustering A clustering where a single document might be placed in several clusters at once – clusters might have intersections with each other.

Sparse matrix A matrix storage format where only the non-zero values in the matrix are stored together with their positions in the matrix.

Stem group Term groups that record words that are conflated together during stemming and a candidate word to represent the group.

Stemming Automatic conflation algorithms that conflate words that share the same word stem.

Stop-list A list of stop-words, that should not be indexed.

Stop-words Words that do not convey any practical information and thus are irrelevant for information retrieval.

Suffix A number of characters added at the end of a word to form a derivative form of the word.

Supremum In mathematics, the supremum of an ordered set is the least element that is greater than or equal to each element of the set. Consequently, it is also referred to as the least upper bound. In lattice terminology this is found using the *join* operation.

SVD Singular Value Decomposition (SVD) is the mathematical matrix decomposition technique that lies at the heart of LSA.

Synergy Synergy is our novel keyword extraction technique based on LSA.

Synonymy The coexistence of several possible words for a given meaning or concept.

Term A word or string of characters extracted from a document.

Term weighting The process of determining non-binary weights for term-document relations.

Term/feature space The multidimensional space spanned by all the terms or features extracted from a document collection. Each document is then represented as a vector in this space.

Term-document matrix The matrix representing the weighted term-document relations. Often a sparse matrix format is used to save storage space.

Trie A trie is a tree structure that stores strings in such a way that there is one (and only one) node for every common prefix.

Two-stage approach The splitting up of the document clustering into two stages: Keyword extraction and keyword-based clustering.

User test A test using the opinion, experience or behaviour of actual users to evaluate a system.

Vector (space) model This information retrieval model uses non-binary term-document weights as weights modelling the relation between documents and terms.

Weight function A mathematical function used to weight term-document relations.

Wikipedia Wikipedia is a Web-based, multi-language, free-content encyclopedia written collaboratively by volunteers. *See* `http://www.wikipedia.org`.

Index

Part II

Appendix

Appendix A

Preliminary Experiments with Synergy

Some Preliminary Tests of the "Synergy" Keyword Extractor

*- Keywords in **bold** are inferred (i.e. not present in the original document)*
- Keywords are presented in descending relevance order

Document 1:

Editions of the Dewey Decimal Classifications
The present study is a history of the DEWEY Decimal Classification. The first edition of the DDC was published in 1876, the eighteenth edition in 1971, and future editions will continue to appear as needed. In spite of the DDC's long and healthy life, however, its full story has never been told. There have been biographies of Dewey that briefly describe his system, but this is the first attempt to provide a detailed history of the work that more than any other has spurred the growth of librarianship in this country and abroad.

Pure Truncation:
dewey, ddc, edition, biographies, decimal, told, story, 1876, abroad, spite

LSA600:
dewey, edition, decimal, history, ddc, 1876, full, country, classification, growth

LSA400:
dewey, edition, decimal, classification, history, 1876, librarianship, growth, full, attempt

LSA300:
dewey, decimal, edition, classification, history, librarianship, **udc**, 1876, growth, attempt

LSA200:
decimal, classification, dewey, edition, librarianship, history, **udc**, published, 1876, detailed

LSA150:
classification, decimal, dewey, edition, librarianship, **udc**, history, **universal**, published, present

LSA100:
classification, decimal, edition, dewey, librarianship, **udc**, history, published, **universal**, scheme

LSA50:
classification, decimal, edition, **udc**, scheme, history, **subject**, **documentation**, dewey, librarianship

LSA40:
classification, decimal, scheme, **udc**, edition, growth, dewey, **special**, **subject**, **universal**

LSA30:
classification, decimal, scheme, **udc**, edition, growth, **knowledge**, **subject**, **general**, **universal**

LSA20:
classification, **subject**, decimal, **documentation**, scheme, **indexing**, edition, **udc**, **international**, **cataloguing**

LSA10:
classification, **subject**, **x**, **library**, **catalog**, **theory**, **language**, **indexing**, **rules**, **general**

LSA5:
library, classification, **book**, **theory**, **study**, **research**, **librarians**, **subject**, **analysis**, **university**

Note: The Universal Decimal Classification (UDC) was originally derived from Dewey Decimal Classification (DDC) and LSA has correctly identified the relationship between these two schemes.

Document 9:

Access to Libraries in College
This study assumed that an additional use study held less promise than an analytical consideration of concepts. The basic approach was a survey comparing traditional and current professional ideas on direct access. Principal data-gathering instruments were documentary analysis and opinion questionnaire.

Findings of the documentary analysis included the following: Research from 1890 to 1970 on the direct shelf approach and browsing left the problems largely unresolved and evidently resistant to established methods of use and user research. The need for an exhaustive study of concepts was confirmed.

Open shelf libraries--organized through shelf classification and relative location--were meant to arouse the intellectual, social, and political interest of the average citizen and affect this democratic self-realization.

Definitions of "browsing" varied greatly: self-indulgence by the untutored in objectionable works; beneficial self-education for the general reader; valuable guidance for the scholar in his research.

Pure Truncation:
shelf, browsing, documentary, realization, indulgence, arouse, confirmed, democratic, direct, promise

LSA600:
shelf, documentary, browsing, direct, democratic, political, concepts, arouse, access, citizen

LSA400:
shelf, documentary, direct, browsing, access, concepts, approach, political, democratic, works

LSA300:
shelf, access, direct, documentary, browsing, concepts, approach, works, reader, research

LSA200:
reader, access, shelf, direct, research, analysis, approach, libraries, gathering, study

LSA150:
research, access, approach, shelf, reader, libraries, concepts, survey, analysis, study

LSA100:
classification, method, research, libraries, concepts, study, college, approach, survey, **area**

LSA50:
classification, libraries, research, **university**, **academic**, data, problems, study, **bases**, survey

LSA40:
classification, libraries, **university**, research, data, **academic**, analysis, study, **books**, survey

LSA30:
classification, libraries, **university**, data, research, **books**, **social**, study, **academic**, survey

LSA20:
libraries, classification, research, **university**, data, **books**, **services**, **academic**, **subject**, survey

LSA10:
libraries, research, **books**, **medical**, **information**, **science**, **librarians**, study, classification, education

LSA5:
libraries, **books**, research, **information**, **librarians**, study, **university**, classification, **public**, problems

Document 422:

On Some Clustering Techniques
The problem of organizing a large mass of data occurs frequently in research.. Normally, some process of generalization is used to compress the data so that it can be analyzed more easily. A primitive step in this process is the "clustering" technique, which involves gathering together similar data into a cluster to permit a significant generalization.

This paper describes a number of methods which make use of IBM 7090 computer programs to do clustering. A medical research problem is used to illustrate and compare these methods.

Pure Truncation:
clustering, generalization, primitive, clusters, occurs, involves, easily, mass, organizing, gathering

LSA600:
clustering, process, data, clusters, methods, generalization, gathering, step, large, problem

LSA400:
process, data, clustering, methods, large, problem, research, medical, clusters, programs

LSA300:
data, process, methods, research, problem, clustering, medical, large, technique, clusters

LSA200:
data, process, problem, research, large, medical, computer, methods, clusters, technique

LSA150:
data, problem, research, large, process, medical, computer, methods, clusters, programs

LSA100:
data, medical, research, large, process, methods, computer, **automatic**, problem, **bases**

LSA50:
data, computer, medical, **automatic**, **document**, large, research, **bases**, methods, technique

LSA40:
data, medical, **document**, large, **automatic**, methods, **bases**, computer, analysis, **management**

LSA30:
data, medical, **automatic**, research, computer, **bases**, **project**, **words**, large, process

LSA20:
data, **automatic**, computer, research, program, **bases**, **system**, **information**, methods, medical

LSA10:
system, **information**, **chemical**, data, **retrieval**, computer, **system**, **search**, **index**, **services**

LSA5:
information, data, **system**, **retrieval**, computer, **chemical**, **search**, **science**, **index**, **bases**

Document 960:

Library Operations Research: A Process of Discovery and Justification
This article begins with a discussion of the broad role of operations research (O.R.) in a society undergoing change. The nature of O.R. terms in a library environment is then considered. The function of models in O.R. is analyzed, the development of a model being contrasted with its formal presentation. Criteria for good models are suggested.. This article then focuses on storage models for libraries, first considering the Dewey classification system from this perspective and then summarizing more current research carried out under the direction of the author with a grant from the National Science Foundation.

Pure Truncation:
models, summarizing, undergoing, focuses, contrasted, justification, begins, grant, discovery, direction

LSA600:
models, operations, article, dewey, storage, role, nature, change, considered, author

LSA400:
models, operations, article, formal, research, classification, terms, role, function, storage

LSA300:
models, operations, article, formal, research, classification, science, change, terms, author

LSA200:
models, operations, science, classification, formal, library, criteria, author, research, system

LSA150:
models, operations, classification, science, library, formal, research, article, criteria, nature

LSA100:
models, classification, library, **project**, science, operations, **theory**, **communication**, process, **selection**

LSA50:
theory, classification, models, library, process, **communication**, **project**, **cost**, **network**, research

LSA40:
theory, models, classification, **project**, **cost**, library, process, research, **decision**, communication

LSA30:
theory, models, **cost**, classification, **project**, library, research, **scientific**, **decision**, science

LSA20:
theory, research, library, **language**, models, **decision**, **project**, **cost**, **analysis**, development

LSA10:
library, **theory**, system, **information**, research, **books**, **scientific**, models, **social**, science

LSA5:
information, library, research, science, **theory**, system, **scientific**, **books**, **social**, classification

Document 989:

An Analysis of the Universal Decimal Classification as a Term System for Nuclear Science and Technology
Explores the possibilities of merging the terminology of the Universal Decimal Classification System with that of a term system - Engineers Point Council's Thesaurus - for nuclear science and technology. Concludes, from the evidence presented, that UDC can be effectively used as a term system. Proposes that the two systems coordinate the terms and merge a major thesaurus (EJC) with an effective classification scheme of international scope (UDC) to provide a needed tool in the area of classification and documentation.

Pure Truncation:
nuclear, udc, merge, decimal, universal, term, classification, proposes, explores, thesaurus

LSA600:
udc, decimal, classification, term, universal, thesaurus, technology, nuclear, scheme, documentation

LSA400:
udc, classification, decimal, universal, term, thesaurus, technology, documentation, scheme, nuclear

LSA300:
classification, udc, decimal, universal, term, thesaurus, technology, documentation, scheme, system

LSA200:
classification, udc, decimal, term, universal, thesaurus, scheme, technology, science, documentation

LSA150:
classification, decimal, udc, term, universal, thesaurus, documentation, technology, science, international

LSA100:
classification, decimal, udc, documentation, term, thesaurus, universal, science, scheme, international

LSA50:
classification, decimal, thesaurus, udc, term, scheme, science, documentation, **information**, **retrieval**

LSA40:
classification, decimal, thesaurus, udc, scheme, **information**, term, **indexing**, documentation, **retrieval**

LSA30:
classification, decimal, thesaurus, scheme, **indexing**, udc, **information**, **retrieval**, **subject**, documentation

LSA20:
classification, **indexing**, **subject**, **retrieval**, decimal, scheme, term, documentation, **information**, **language**

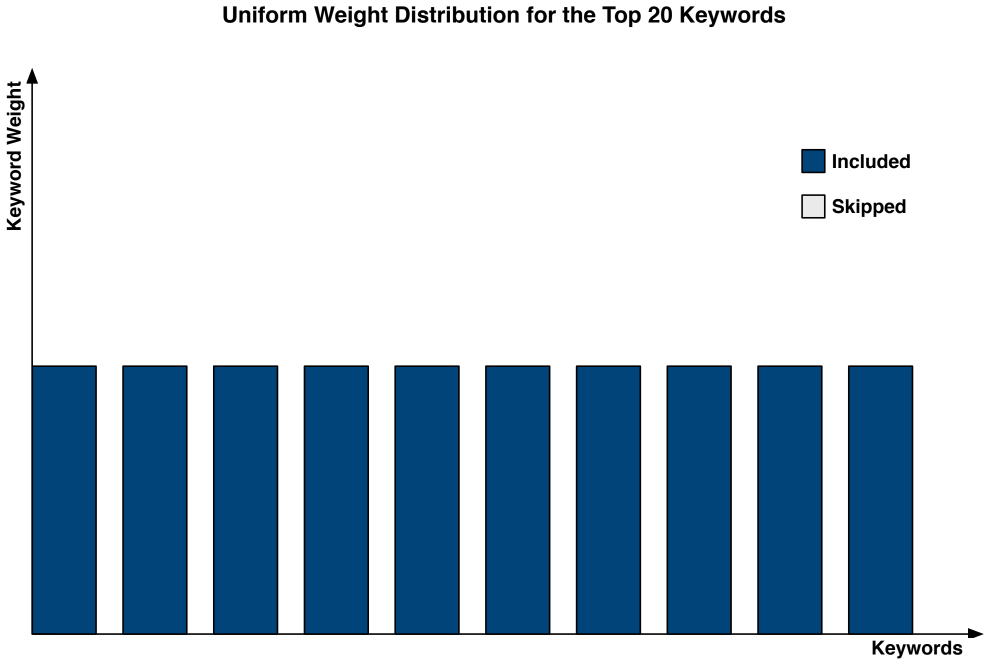
LSA10:
classification, **indexing**, **language**, **automatic**, **theory**, **retrieval**, **subject**, **document**, **information**, **book**

LSA5:
retrieval, classification, **indexing**, **document**, **theory**, system, **automatic**, **language**, term, information

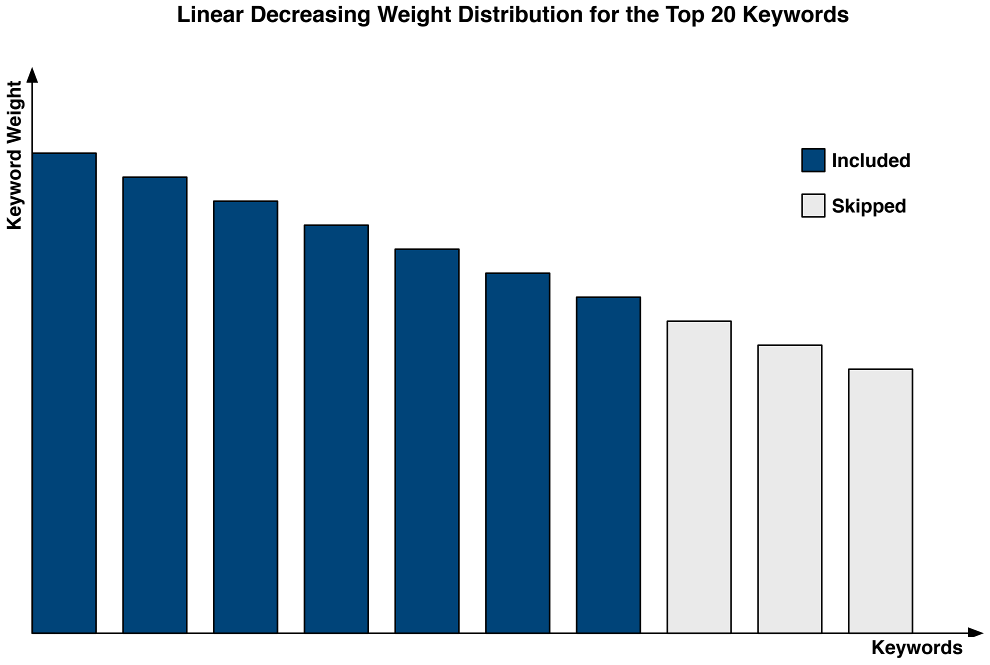
Appendix B

Illustrations of Keyword Cutoff Scheme

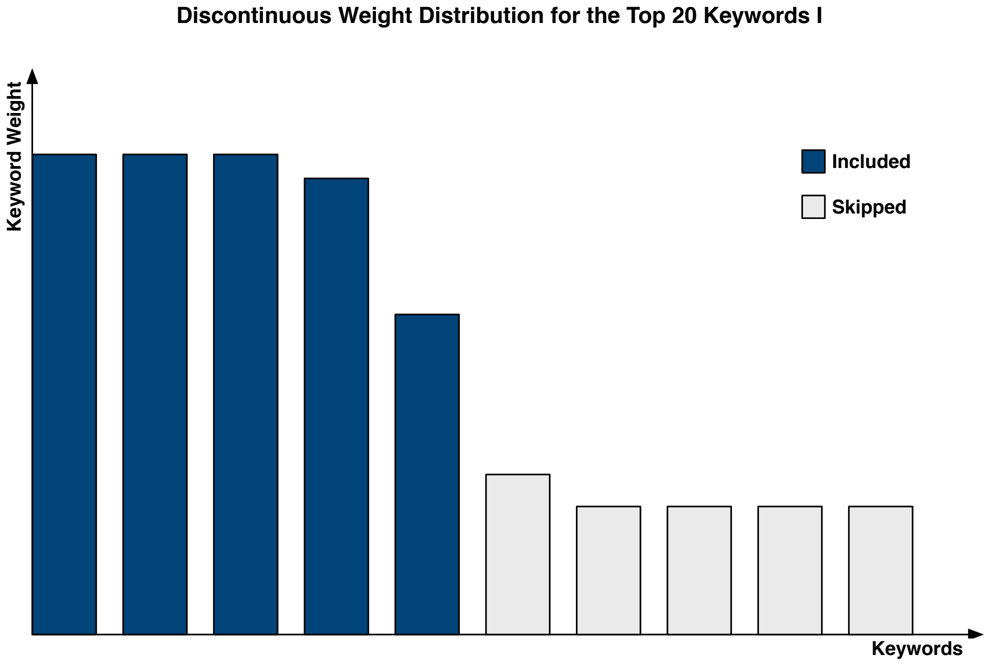
B.1 Uniform Weight Distribution



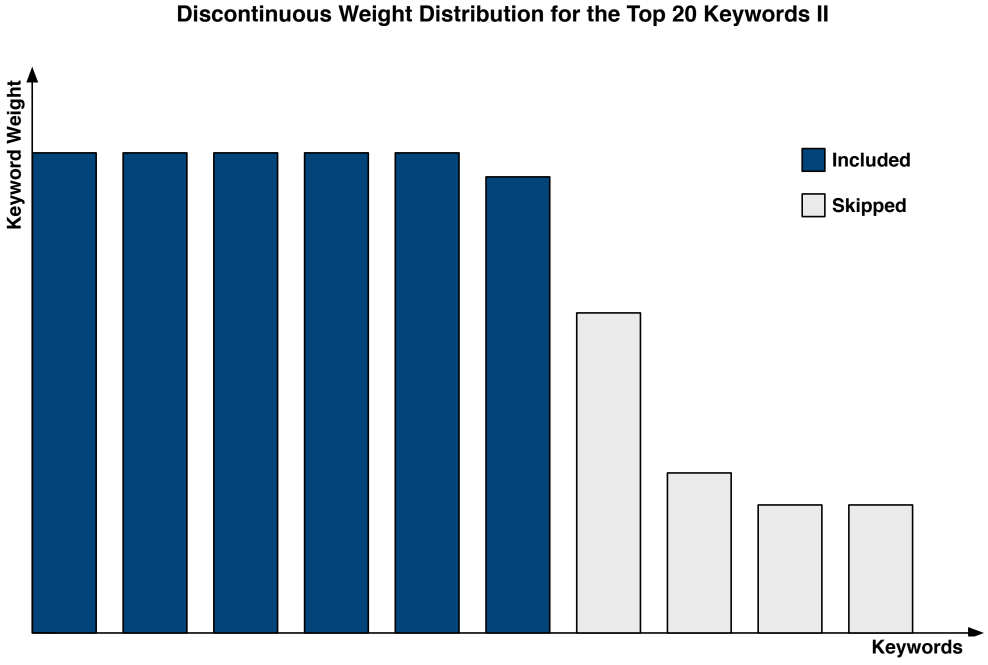
B.2 Linear Decreasing Weight Distribution



B.3 Discontinuous Weight Distribution I



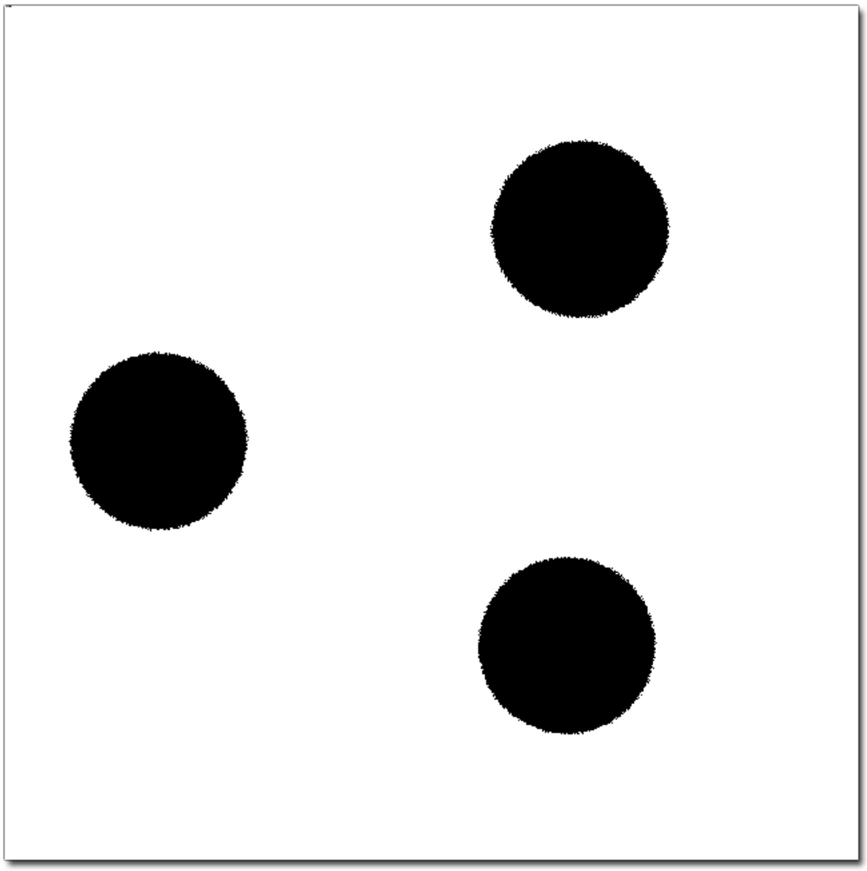
B.4 Discontinuous Weight Distribution II



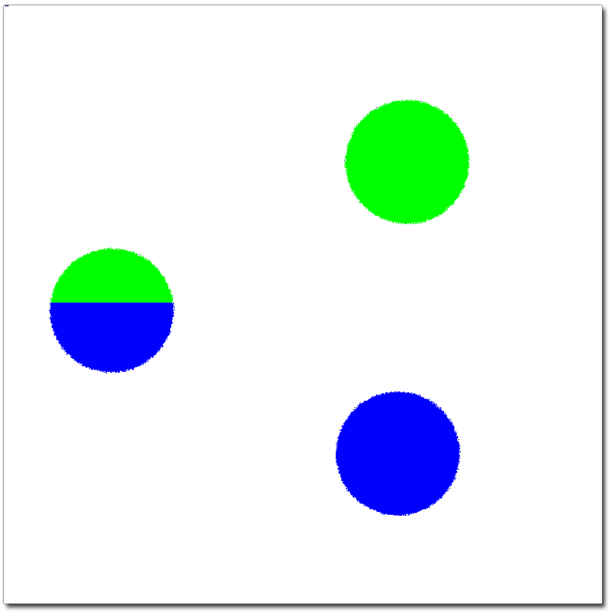
Appendix C

Demonstration of Bisection

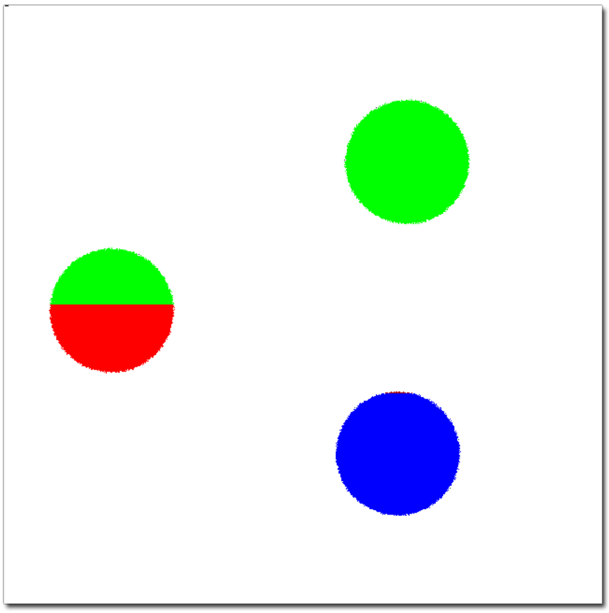
C.1 The Initial Document Set



C.2 The Document Set After the First Partitioning



C.3 The Document Set After the Second Partitioning



Appendix D

Chernoff Bounds for CURE

In [GRS98], the following equation for calculating sample sizes is derived using Chernoff Bounds:

$$s_{min} = \xi k\rho + k\rho * log(\frac{1}{\delta}) + k\rho\sqrt{(log(\frac{1}{\delta})^2 + 2\xi log(\frac{1}{\delta})} \tag{D.1}$$

s_{min} is the minimum sample size we can make, while at the same time ensuring that no clusters will be missed during clustering.

ξ is the number of data points that should represent the smallest cluster.

k is the number of clusters desired.

ρ is the inverse of the ratio between the smallest cluster and the average cluster (e.g. if ρ is 4, then the size of the smallest cluster is 25% of the average cluster).

δ is the desired probability that the sample contains at least $f|u|$ points from any cluster u, where $f = \frac{\xi}{u_{min}}$.

Example, 10,000 documents:
If we assume the ideal case, where all clusters are dense and well-spread, and desire that all clusters be of size 10 (because of more-like-this) when clustering completes, we thus need $k = \frac{10000}{10} = 1000$. We set $\xi = 5$ because we need a lot of data points to represent the cluster in a multi-dimensional space (5 might even be too few). If we assume that all clusters naturally are of approximately the same size, we can set $\rho = 1$. And finally, we set $\delta = 0.01$, since we don't want to miss any clusters.

Inserting the numbers into the formula:

$$s_{min} = 5*1000*1+1000*1*log(\frac{1}{0.01})+1000*1\sqrt{(log(\frac{1}{0.01})^2 + 2*10*log(\frac{1}{0.01})} = 13633 \tag{D.2}$$

We see that if we want to use sampling, we actually need a sample *larger* than the amount of available data points!

Appendix E

Applied Hash Functions

E.1 h1 Based on Thomas Wang's 32 bit Mix Function

```
unsigned int hash_h1(unsigned int key, unsigned int size) //Size should be power of 2
{

    key += ~(key << 15); //Thomas Wang's 32 bit Mix Function
    key ^=  (key >> 10);
    key +=  (key << 3);
    key ^=  (key >> 6);
    key += ~(key << 11);
    key ^=  (key >> 16);

    return key % size;
}
```

E.2 h2 Based on Robert Jenkin's 32 bit Mix Function

```
unsigned int hash_h2(unsigned int key, unsigned int size) //Size should be power of 2
{
    key += (key << 12); //Robert Jenkins' 32 bit Mix Function
    key ^= (key >> 22);
    key += (key << 4);
    key ^= (key >> 9);
    key += (key << 10);
    key ^= (key >> 2);
    key += (key << 7);
    key ^= (key >> 12);

    if((key % 2) == 0) //Always return an odd number, since size is even
        return (key-1) % size;

    if(key != 0)
        return key % size;
    else
        return 1;
}
```

Appendix F

Clustering Toolkit Settings

F.1 Filtering

- Minimum document frequency for terms: 10
- Maximum document frequency for terms: 50 %
- Remove 1-character terms
- Remove numbers with less than 4 digits

F.2 Bigram Extraction

- Extract bigrams using scheme 2 (based on previous lattice clustering)
- Minimum document frequency: 10
- Minimum global frequency: 10

F.3 Weighting

- Local weighting algorithm: Mondosoft's weighting scheme
- Global weighting algorithm: Log-Entropy
- Bigram weights: 42

F.4 Keyword Extraction

- Keyword extraction algorithm: Synergy
- LSA dimension: 300
- Normalise document vectors
- Minimum keywords per page: 10
- Maximum keywords per page: 20
- Cut-off factor: 0.5

F.5 Clustering Algorithm Specific Settings

- Apriori minimum support: 5
- Hierarchy-based algorithm distance measure: Cosine
- Number of partitionings for divisive algorithm: 90%

Appendix G

Extracted Bigrams

G.1 Scheme 1: Bigrams found for MuggleNet

ROBBIE FISCHER
ROBERT LANTO
HALF BLOOD
SEVERUS SNAPE
JULIA EISENBERG
VIKTOR KRUM
DEATH EATERS
JOSH SMITH
EISENBERG SPHAERA
FRED GEORGE
SAID SPANKY
ANDREW LEE
LORD VOLDEMORT
ALBUS DUMBLEDORE
EMMA RUPERT
RIDDLE TOM
TOM RIDDLE
JAMES SIRIUS
WIZARD WORLD
FAIRY TALE
PETER PETTIGREW
JAMES LILY
UNITED STATES
DANIEL RADCLIFFE
EMMA WATSON
CHO CHANG
HOGWARTS EXPRESS
GINNY WEASLEY
TOM FELTON
GARY OLDMAN
LEAKY CAULDRON
JASON ISAACS
LAVENDER BROWN
HARRY POTTER
QUIDDITCH GAME
TIME TURNER
SIRIUS BLACK
DEVON MURRAY
BOOK 6
PANSY PARKINSON
DRACO MALFOY
PARVATI PATIL
SEAN BIGGERSTAFF
ALAN RICKMAN
HERMIONE GRANGER
RUPERT GRINT
AUNT PETUNIA
CEDRIC DIGGORY
GILDEROY LOCKHART
NEWS ARCHIVE
SORTING HAT
TOP SITES
SIXTH BOOK
BOOK SIX
MIKE NEWELL
OLIVER PHELPS
POA TRAILER
DAN RADCLIFFE
BARTY CROUCH
FRED WEASLEY
OFFICIAL SITE
CHRISTIAN COULSON
REMUS LUPIN
HARRYPOTTER COM
MARCUS FLINT
NUMBER 12
BOOK 5
CAPTION CONTEST
RON WEASLEY
PART 3
RELEASE DATE
JAMES PHELPS
MAGGIE SMITH
MICHAEL GAMBON
RICHARD HARRIS
JULIE WALTERS
JOHN CLEESE
CHRIS COLUMBUS
WARWICK DAVIS
ROBBIE COLTRANE
RICHARD GRIFFITHS
HERMIONE RON
HARRY HERMIONE
FIONA SHAW
KENNETH BRANAGH
JAMES POTTER
VINCENT CRABBE
GREGORY GOYLE
BABY HARRY
STAR WARS
BOOK FIVE
BONNIE WRIGHT
MOVIE III
COME TOGETHER
MARK EVANS
FAN FICTION
OLIVER WOOD
EMAIL ADDRESS
DARK MARK
BOOK TROLLEY
SUSAN BONES
RON HERMIONE
PROFESSOR LUPIN
GALADRIEL WATERS
NAME ORIGINS
EMMA THOMPSON
MOANING MYRTLE
DIE RON
TERENCE HIGGS
PROFESSOR TRELAWNEY
QUIDDITCH CUP
MATTHEW LEWIS
LUNA LOVEGOOD
LEAVESDEN STUDIOS

```
TRIWIZARD TOURNAMENT
CHRIS RANKIN
BELLATRIX LESTRANGE
LUCIUS MALFOY
MINERVA MCGONAGALL
FLEUR DELACOUR
DIAGON ALLEY
MADAME MAXIME
DEAN THOMAS
MUGGLENET INTERACTIVE
AVADA KEDAVRA
CAST MEMBERS
MESSAGE BOARDS
GEORGE WEASLEY
MAGICAL CREATURES
STAN SHUNPIKE
WHOMPING WILLOW
BOOK FOUR
BOOK 1
JK ROWLING
ARTHUR WEASLEY
BOOK 7
JUSTIN FINCHFLETCHLEY
HOGWARTS HOUSES
CHOCOLATE FROG
JOB OPENINGS
DAILY PROPHET
NYMPHADORA TONKS
ANGELINA JOHNSON
PROFESSOR SNAPE
FAN ART
MAD EYE
NARCISSA MALFOY
CHARACTER BIOGRAPHIES
NEXT BOOK
MIDDLE NAME
MAIN PAGE
HOUSE ELF
QUIDDITCH TEAMS
CBBC NEWSROUND
ENTIRE BOOK
REAL MAGIC
CHAPTER PICTURES
ARGUS FILCH
RONALD WEASLEY
FOURTH MOVIE
RELEASE DATES
NICHOLAS FLAMEL
MUGGLE WORLD
NICOLAS FLAMEL
LOOK LIKE
TIME TRAVEL
FAN CLUB
PROFESSOR BINNS
NEVILLE LONGBOTTOM
LILY POTTER
DARK FOREST
DUDLEY DURSLEY
BOOK 3
BOOK THREE
LEE JORDAN
PRIVET DRIVE
VERNON DURSLEY
HOUSE CUP
SHRIEKING SHACK
BOOK 4
LILY EVANS
GODRIC GRYFFINDOR
MUGGLENET CHAT
POTTER PROFILES
PROFESSOR FLITWICK
PROFESSOR QUIRRELL
WORLD CUP
LLOYD ALEXANDER
BOOK COVERS
MOVIE IV
TEASER TRAILER
ZOE WANAMAKER
KATIE BELL
FEEDBACK FORM
ALFONSO CUARON
YULE BALL
SEAMUS FINNIGAN
DAVID THEWLIS
AZKABAN MOVIE
BOOK SEVEN
PERCY WEASLEY
MOLLY WEASLEY
CHAPTER 1
LOVE TRIANGLE
MUNDUNGUS FLETCHER
DEATH CURSE
KNIGHT BUS
FAT LADY
INVISIBILITY CLOAK
DEATH EATER
MARAUDERS MAP
FLOO POWDER
LUDO BAGMAN
POLL RESULTS
WEASLEY TWINS
DUELING CLUB
NEWS ARCHIVES
JOKE SHOP
QUIDDITCH MATCH
LIGHTNING BOLT
HOUSE POINTS
PROFESSOR MCGONAGALL
OWL POST
RUBEUS HAGRID
SONG PARODIES
HOGWARTS CASTLE
NEWT SCAMANDER
ARABELLA FIGG
MRS WEASLEY
MAIN CHARACTER
COMMON ROOM
AUNT MARGE
SET REPORT
VIDEO GAMES
HOUSE ELVES
VICTOR KRUM
CHOCOLATE FROGS
GRYFFINDOR HOUSE
HP STORIES
RITA SKEETER
PROFESSOR DUMBLEDORE
FLYING CAR
HOGWARTS STUDENTS
HOGWARTS STAFF
MADEYE MOODY
WEASLEY FAMILY
FORD ANGLIA
DIANE DUANE
GREEK MYTHOLOGY
NORTH TOWER
HEAD BOY
KING ARTHUR
MRS NORRIS
OAK PARK
FOUR FOUNDERS
MOVIE CAST
GOLDEN SNITCH
PRIORI INCANTATEM
SALAZAR SLYTHERIN
FAN SITES
JAMIE WAYLETT
COS FORUMS
BOOK 2
ORDER MEMBERS
HOGWARTS SCHOOL
MAIN CHARACTERS
FIRST MOVIE
POLYJUICE POTION
PATRONUS CHARM
FIRST BOOK
WORLD WAR
MADAM MAXIME
FIFTH BOOK
ALICIA SPINNET
HARRY GINNY
HATE MAIL
BOOKMARK SITE
JOHN WILLIAMS
CHARACTER DEVELOPMENT
FORBIDDEN FOREST
FOURTH FILM
SORTING CEREMONY
RECENT UPDATES
QUIDDITCH PLAYERS
IMPERIOUS CURSE
CORNELIUS FUDGE
POTTER PROFILE
ONLINE CHAT
THIRD MOVIE
STEPHEN FRY
DOLORES UMBRIDGE
HARMIONE GRANGER
PHOTOSHOP FUN
TRADING CARD
DANRADCLIFFE COM
CHAPTER 13
PROFESSOR UMBRIDGE
```

ERNIE MACMILLAN
GRYFFINDOR TOWER
PREVIOUS POLLS
FIRST TASK
SECOND TASK
POTTER HARRY
HP GALLERIES
STEVE KLOVES
PENELOPE CLEARWATER
MR OLLIVANDER
TRUE LOVE
TRIWIZARD CUP
5TH BOOK
DAVID HEYMAN
GREAT HALL
ALASTOR MOODY
SLYTHERIN HOUSE
FANTASTIC BEASTS
FAMILY TREE
PROFESSOR QUIRREL
DARK ARTS
FUTURE BOOKS
PUDDLEMERE UNITED
FOUR HOUSES
COLIN CREEVEY
HUFFLEPUFF HOUSE
MADAM POMFREY
BLOODY BARON
KNOCKTURN ALLEY
BOOK DAY
PROFESSOR SPROUT
CRYSTAL BALL
TEA LEAVES
WORLD BOOK
WARNER BROTHERS
JKR SAID
THREE BROOMSTICKS
FLOO NETWORK
MARY GRANDPRE
VIDEO GAME
HARRY CHO
CHAT TRANSCRIPT
USA TODAY
NO DOUBT
WANDLESS MAGIC
HUNGARIAN HORNTAIL
WARNER BROS
CLASSICAL MUSIC
THIRD BOOK
FUNNY EXCERPTS
CHESS GAME
AZKABAN PRISON
MR WEASLEY
SECRET KEEPER
FIDELIUS CHARM
MUGGLE STUDIES
QUIDDITCH PITCH
COMIC RELIEF
UNCLE VERNON
MRS FIGG
WITCHING HOUR
HERMIONE HARRY
PHINEAS NIGELLUS
DARK SIDE
LAST BOOK
RED HERRING
FOUR CHILDREN
UNFORGIVABLE CURSES
MR MASON
CARD GAME
SHRUNKEN HEAD
BAD PEOPLE
IMPERIUS CURSE
FIRST CHAPTER
BLACK FAMILY
MORAL FIBER
ENTERTAINMENT WEEKLY
GREEN EYES
GRIMMAULD PLACE
THIRD TASK
FORGOT PASSWORD
COMING SOON
HEADLESS NICK
PLAY QUIDDITCH
SEVENTH BOOK
LOST PROPHECY
MAGIC QUILL
JKROWLING COM
QUIDDITCH CAPTAIN
MAY 10
MAY 12
FANTASY WORLD
QUIDDITCH TEAM
HARRY LUNA
MUGGLENET COM
HARRY RON
PROF MCGONAGALL
FAT FRIAR
RAVENCLAW HOUSE
OFFICIAL WEBSITE
JO ROWLING
CONTACT DAN
VOLDEMORT GOBLET
J K
HOSPITAL WING
SPECIAL EFFECTS
CHANGELING HYPOTHESIS
GRYFFINDOR QUIDDITCH
PROFESSOR KETTLEBURN
SECOND WAR
JKR COM
WEIRD SISTERS
COVER ART
DAN EMMA
UNOFFICIAL GUIDE
RADCLIFFE INTERVIEW
H HR
FUN STUFF
BOOK TWO
SHORT STORIES
PETUNIA DURSLEY
HOLY GRAIL
ZACHARIAS SMITH
ATTEND HOGWARTS
PURE BLOOD
TV SHOWS
GOF PIC
POTIONS CLASS
LITTLE PEOPLE
FROG CARDS
DADA TEACHER
UNICORN BLOOD
FIRST YEARS
RON DIE
FRANK BRYCE
LEVEL NINE
DARK FORCES
ULTIMATE UNOFFICIAL
UNFORGIVABLE CURSE
HOGWARTS HOUSE
WEB SITE
DARK MAGIC
CONSTANT VIGILANCE
DIVINATION TEACHER
FAN SITE
LICENSE PLATES
BLOOD PRINCE
FOURTH BOOK
HALFBLOOD PRINCE
SS PS
TOO OLD
6 4
NITWIT BLUBBER
LOCKED DOOR
RIDDLE HOUSE
BAD GUYS
KILLING CURSE
AMERICAN VERSION
JOSHUA SMITH
SCARLET WOMAN
CLUES DISCUSSION
MRS DURSLEY
K ROWLING
COS MOVIE
WIZARDING WORLD
NEARLY HEADLESS
PRESS CONFERENCE
DARK WIZARDS
CROUCH JR
HBP RELEASE
DARK CREATURES
SIRIUS DIED
SIXTH YEAR
RED HEN
BERTHA JORKINS
EVIL PEOPLE
MUGGLENET STAFF
MALINE FREDN
DELUXE EDITION
SOMEONE ELSE
BOOK FESTIVAL
GOF VOLDEMORT
CHAPTER 29

HAPPY BIRTHDAY
CHAPTER 12
10 MINUTES
GIANT SQUID
KILL HARRY
FUTURE BOOK
RON HARRY
JKR ROWLING
HOGWARTS STUDENT
MAGICAL ABILITY
PLOT THICKENS
MR CROUCH
QUESTION MARK
IMPOSTER MOODY
DADA TEACHERS
FIRST DAY
BOB SINDELDECKER
BROWN HAIR
BOOK SERIES
BRANDON FORD
PROF FLITWICK
LOST DAY
PROJECT LEGILIMENSIA
WORST MEMORY
HP MOVIES
WINDOWS MEDIA
UNDERGROUND LAKE
CHAPTER NINE
DARK HAIR
HOGWARTS INFORMATION
RED HAIR
CHAPTER 6
DANIELA TEO
CRUCIATUS CURSE
FUN LISTS
WELCOME BACK
PHOENIX FILES
GOF CLIPS
PS SS
STUPID ABOU
VISUAL EFFECTS
JULY 16
WHY SNAPE
HBP COVERS
LEAST FAVORITE
SCHOOL HOUSE
BOOKS 6
BEST FRIEND
FRONT PAGE
MESSENGER STUFF
LITTLE THINGS
AMAZON COM
MEMORY CHARMS
TERRY PRATCHETT
SCIENCE FICTION
DAN HOPPEL
HEY HARRY
BIG NEWS
NEXT MINISTER
STAR TREK
HAPPY ENDING
FAVORITE BOOK
ESSENCE DIVIDED
MEMORY CHARM
YOUNG WIZARDS
SCHOOL DAYS
LONDON UNDERGROUND
TEA COZY
FAVORITE QUOTES
SILVER HAND
PROMOTIONAL POSTER
BLACK PARK
TOM MARVOLO
RED HERRINGS
POTTER MOVIE
SPOILER WARNING
FINAL BATTLE
RIGHT HAND
HBP DELUXE
HP FAN
FIRST YEAR
INNER EYE
THRICE DEFIED
FULL MOON
CHAPTER 27

G.2 Scheme 2: Bigrams found for Wikipedia

SQUARE KILOMETRES
STATES DEPARTMENT
ALKALI METAL
KINETIC ENERGY
ISAAC NEWTON
ACETIC ACID
SPACECRAFT PROPULSION
HEAT CAPACITY
PLANCK'S CONSTANT
MOLAR VOLUME
BRONZE AGE
DANCE MUSIC
BYZANTINE EMPIRE
DEMAND CURVE
GREGORIAN CALENDAR
SOIL EROSION
RELIGIOUS FUNDAMENTALISTS
GEOCENTRIC MODEL
BINARY STAR
MISSISSIPPI RIVER
DIESEL ENGINE
SAN DIEGO
DIEGO CALIFORNIA
SUMMER SOLSTICE
FOOTBALL TEAM
LEAGUE TEAM
20TH CENTURY
CENTURY 20TH
PLANET VENUS
INTERSTELLAR MEDIUM
SEPTEMBER OCTOBER
5TH CENTURY
SEDIMENTARY ROCK
AEROSPACE ENGINEERING
NEOCLASSICAL ECONOMISTS
PHOTOGRAPHIC FILM
INFERIOR PLANET
CENTRAL BANK
TURING MACHINE
OX AC
AUSTRIAN SCHOOL
STATES FEDERAL
LAW JURISDICTIONS
SATELLITE NAVIGATION
TEMPERATE CLIMATE
XRAY CRYSTALLOGRAPHY
COVALENT BOND
CASE LAW
ATOMIC NUCLEUS
NITRIC ACID
NATIVE AMERICAN
1ST CENTURY
EASTERN EUROPE
1450S 1460S
MEDITERRANEAN SEA
INDIANA JONES
LEAP YEARS
MARS JUPITER
FM RADIO
FINANCE TOPICS
VASCULAR PLANTS
SIDEREAL TIME
MARGINAL UTILITY
VAPOR PRESSURE
GRAMPOSITIVE BACTERIA
COPYRIGHT PROTECTION
NUCLEAR FUSION
PEOPLE'S REPUBLIC
COMBUSTION ENGINE
MIGHT OTHERWISE
VARIABLE STAR
FREE TRADE
JEWISH BIBLE
CHRISTIAN BIBLE
PLATE TECTONICS
SPOKEN LANGUAGE
LANGUAGE SPOKEN
THOMAS JEFFERSON

EUCLIDEAN GEOMETRY
SECONDARY EDUCATION
SECONDARY SCHOOL
SOUTHERN HEMISPHERE
COURT JUSTICE
HELIOCENTRIC MODEL
NUCLEAR FISSION
2ND MILLENNIUM
BC 2ND
2ND CENTURY
EMPEROR HIROHITO
NORTHERN HEMISPHERE
BASIC TOPICS
3RD MILLENNIUM
3RD CENTURY
LATIN WORD
YEARS AGO
PAUL II
POPE PAUL
HEALTH CARE
MEDICAL CARE
LOCAL FOOD
LOS ANGELES
CALIFORNIA LOS
DEVELOPMENTAL BIOLOGY
SUPERNOVA EXPLOSION
ASP BOOKID
BOOK ASP
KING JAMES
NEWTON'S LAW
EXTRATERRESTRIAL LIFE
INTERNATIONAL AGREEMENT
TRADE AGREEMENT
SIR WILLIAM
SIR JOHN
SUPREME COURT
REFORM MOVEMENT
AGRICULTURAL POLICY
1460S 1470S
CIMT DICTUNIT
GAIA HYPOTHESIS
MOTION PICTURE
NAZI GERMANY
NAZI WAR
1850S 1860S
MILLENNIUM BC
POTASSIUM NITRATE
PRICE ELASTICITY
AMERICAN TRIBES
SAMESEX MARRIAGE
SWEDEN PRIZE
JUNE JULY
FOOTBALL LEAGUE
TERTIARY EDUCATION
STEAM ENGINE
LINEAR ALGEBRA
RIGHT SIDE
MICROWAVE BACKGROUND
COSMIC BACKGROUND
BACKGROUND RADIATION
LIVING ORGANISMS
OZONE LAYER
EARTH'S ATMOSPHERE
OTTOMAN EMPIRE
CLIMATE CHANGE
CHEMICAL EQUILIBRIUM
SPACE PROBE
HTML HTTP
SOUTH POLE
NORTH POLE
RECREATIONAL DRUGS
ANGELES CALIFORNIA
MOLECULAR WEIGHT
ATOMIC WEIGHT
AC UK
UK HTTP
LUNAR ECLIPSE
SOLAR ECLIPSE
TELEPHONE SERVICE
TELEPHONE NETWORK
1470S 1480S
NAPOLEON BONAPARTE
GAMMA RAY
GAMMA RADIATION
APPARENT MAGNITUDE
STAR APPARENT
DEMOCRATIC PARTY
FIXED POINT
POPE PIUS
ABACCI COM
WWW ABACCI
ELECTROMAGNETIC SPECTRUM
ENERGY SPECTRUM
SPIRAL GALAXIES
CARBON DIOXIDE
COSMIC MICROWAVE
MICROWAVE RADIATION
ORTHODOX CHURCH
DIFFERENTIAL EQUATION
INTEREST RATE
SEXUAL REPRODUCTION
DARK MATTER
ELECTRON SHELL
DNA REPLICATION
GENDER ROLE
DE MEDICI
TAX TARIFF
FIRST DAY
FIRST PRESIDENT
FIRST UNITED
HUMAN ANATOMY
OCTOBER NOVEMBER
ENGLAND SCOTLAND
QUEBEC CITY
DIRECTOR WRITER
WRITER DIRECTOR
MOVIE DIRECTOR
FILM DIRECTOR
ACTOR DIRECTOR
INDIVIDUAL CAPITAL
RAIL TRANSPORT
CHARLES DARWIN'S
DARWIN'S THEORY
RUSSIAN REVOLUTION
SEXUAL BEHAVIOR
GROWTH HORMONE
WASTE WATER
SCIENCE RESEARCH
RACE CAR
GREEK ALPHABET
SALT LAKE
SALT WATER
GLOBAL CLIMATE
EVOLUTIONARY BIOLOGY
EVOLUTIONARY THEORY
ELECTRIC CURRENT
JESUS CHRIST
CHRIST CHURCH
TELEVISION SHOW
RADIO SHOW
SOUTH PACIFIC
PACIFIC OCEAN
IRON AGE
GREEK LETTER
GREAT BRITAIN
TERM USED
MIGHT WANT
1480S 1490S
MARKET SHARE
NUCLEAR REACTOR
HOMO SAPIENS
WESTERN EUROPE
WESTERN EUROPEAN
WESTERN COUNTRIES
WESTERN SOCIETY
WESTERN CULTURE
LABOR THEORY
1840S 1850S
WILLIAM HENRY
JOHN WILLIAM
KING WILLIAM
BERLIN WALL
BERLIN GERMANY
CONGO REPUBLIC
RED DWARF
WHITE DWARF
DWARF STAR
DNA STRANDS
18TH CENTURY
CENTURY 18TH
HUMAN EAR
COLLEGE CAMBRIDGE
CAMBRIDGE UNIVERSITY
MEXICO CITY
RADIOACTIVE DECAY
PLANET MARS
EARTH MARS
1820S 1830S
1830S 1840S
SAN FRANCISCO
FRANCISCO CALIFORNIA
COMMUNIST CHINA
SOVIET COMMUNIST

COMMUNIST PARTY
COMMUNIST GOVERNMENT
TELEVISION STATION
RADIO STATION
LABOUR PARTY
NITROGEN OXYGEN
NERVE CELLS
1810S 1820S
DIGITAL SIGNAL
DIGITAL COMPUTER
BRIGHTEST STAR
LAND AREA
EUROPEAN COLONIZATION
TRADEMARK LAW
MARGINAL COST
HEALTH SCIENCE
EAST INDIA
SOUTH EAST
NORTH EAST
EAST AFRICA
1800S 1810S
1790S 1800S
MATH MATHBF
GERMAN ARMY
BRITISH ARMY
FRENCH ARMY
VECTOR FIELD
VECTOR SPACE
TELEVISION BROADCAST
RADIO BROADCAST
INFRARED SPECTROSCOPY
INFRARED RADIATION
INFRARED LIGHT
BIG BANG
BANG THEORY
REAL NUMBER
INDIA COMPANY
BRITISH INDIA
SOUTH AFRICAN
AFRICAN REPUBLIC
1490S 1500S
CLASSICAL LIBERAL
CLASSICAL GREEK
CLASSICAL MUSIC
COMMONWEALTH COUNTRIES
BRITISH COMMONWEALTH
SYMBOL NUMBER
NAME SYMBOL
RADIOACTIVE ISOTOPES
RADIOACTIVE ELEMENTS
DIVISION PLANTS
1630S 1640S
1620S 1630S
KING HENRY
SCIENCE FICTION
AIR BALLOON
AIR PRESSURE
AIR TRANSPORT
COSMIC RAY
1640S 1650S
SQUARE METRES

MATH FRAC
STOCK OPTION
1600S 1610S
1590S 1600S
1650S 1660S
1660S 1670S
RED ALGAE
GREEN ALGAE
ROAD TRANSPORT
1610S 1620S
INDUS VALLEY
INDUS RIVER
TELEVISION SERIES
1500S 1510S
1670S 1680S
DISAMBIGUATION PAGE
1580S 1590S
SKY OBJECTS
1680S 1690S
1690S 1700S
MEDIAN INCOME
MEDIAN AGE
RED COLOR
RED GREEN
RED BLOOD
RED LIGHT
GENUS HOMO
MEDICAL TREATMENT
CANCER TREATMENT
FOSSIL FUEL
1510S 1520S
1570S 1580S
PREFECTURE CAPITAL
MILITARY POWER
MILITARY HISTORY
HISTORY MILITARY
MILITARY FORCE
BIOLOGICAL WEAPONS
BIOLOGICAL EVOLUTION
BIOLOGICAL CELLS
FORMULA CH
CHEMICAL FORMULA
1720S 1730S
1710S 1720S
COGNITIVE PSYCHOLOGY
COGNITIVE SCIENCE
COPYRIGHT LAW
GENOME PROJECT
HUMAN GENOME
1550S 1560S
1540S 1550S
1520S 1530S
1560S 1570S
1770S 1780S
1780S 1790S
1760S 1770S
HUMAN BRAIN
KING LOUIS
LOUIS DE
STONE AGE
WORLD EUROPE

CENTURY EUROPE
1750S 1760S
1700S 1710S
SOLAR WIND
1740S 1750S
1730S 1740S
SPACE MISSION
SOLAR NEBULA
NORTH WEST
WEST GERMANY
PERSON MAY
1530S 1540S
CATHOLIC CHURCH
ROMAN CATHOLIC
AMINO ACID
15TH CENTURY
CENTURY 15TH
ELECTROMAGNETIC WAVE
ELECTROMAGNETIC RADIATION
ELECTROMAGNETIC FIELD
LIBERAL PARTY
ELECTRIC CHARGE
ELECTRON CHARGE
NUCLEAR WEAPONS
CHEMICAL WEAPONS
CHAIN REACTION
ANCIENT GREECE
STABLE ISOTOPES
ANCIENT EGYPTIAN
DE PARIS
PARIS FRANCE
WHITE BLOOD
WHITE LIGHT
BLACK HOLE
CHILD DEVELOPMENT
QUANTUM MECHANICS
COUNTY CALIFORNIA
CHARLES DARWIN
BRAND MANAGEMENT
BRAND NAME
MENTAL EVENTS
HEART DISEASE
MAIN ARTICLE
17TH CENTURY
CENTURY 17TH
DIGESTIVE SYSTEM
CHARLES II
II GERMANY
II KING
II GERMAN
II UNITED
WAR II
RIVER VALLEY
ROYAL COLLEGE
ROYAL SOCIETY
BRITISH ROYAL
ROYAL FAMILY
QUANTUM GRAVITY
NUCLEAR FUEL
EUROPEAN UNION
EUROPEAN COUNTRIES

EUROPEAN COMMUNITY
FRANCE ITALY
PROJECT MANAGEMENT
PROJECT HTTP
OPTICAL FIBER
DNA RNA
WORLD COUNTRIES
WORLD HTTP
WORLD WAR
AMERICAN WRITER
BRITISH WRITER
FRENCH WRITER
ACTOR WRITER
MEDICAL DOCTOR
NATURAL RESOURCES
GENETIC RESOURCES
NATURAL SELECTION
DE LA
FRENCH DPARTEMENT
COM HTTP
RUSSIAN GOVERNMENT
MOLECULAR BIOLOGY
MOLECULAR EVOLUTION
MOLECULAR MASS
GRAVITATIONAL CONSTANT
GRAVITATIONAL FIELD
GRAVITATIONAL FORCE
16TH CENTURY
CENTURY 16TH
SKIN COLOR
SET THEORY
FINANCIAL MARKET
FINANCIAL CAPITAL
RADIO TELESCOPE
BODY CELLS
HUMAN BODY
RELIGIOUS BELIEF
RELIGIOUS MOVEMENT
RELIGIOUS LIFE
COLLEGE OXFORD
OXFORD UNIVERSITY
PHYSICAL ANTHROPOLOGY
PRODUCTION PROCESS
LONDON ENGLAND
ARTICLE GEOGRAPHY
ARTICLE TOPICS
ARTICLE ECONOMY
ARTICLE HISTORY
ARTICLE CULTURE
BRITISH MUSEUM
ENVIRONMENTAL MOVEMENT
SCIENTIFIC CLASSIFICATION
SCIENTIFIC METHOD
SCIENTIFIC THEORY
GAS CONSTANT
PHYSICAL CONSTANT
RADIO GALAXIES
HUMAN KNOWLEDGE
ARAB LEAGUE
SOLID LIQUID
CAUSE CANCER
LAND WATER
ETHNIC GROUP
APRIL MAY
MARCH APRIL
ELECTED GOVERNMENT
ELECTED PRESIDENT
PRESIDENT ELECTED
IONIZATION POTENTIAL
POTENTIAL ENERGY
HYDROGEN BOND
HYDROGEN GAS
SOUTH AMERICA
SOUTH AFRICA
SOUTH SEA
MORAL CODE
MORAL PHILOSOPHY
HUMAN SEXUAL
BLOOD PRESSURE
TEMPERATURE PRESSURE
ANCIENT EGYPT
ANCIENT GREEK
ANCIENT ROMAN
ANCIENT HISTORY
ANCIENT NAME
ANCIENT CITY
PHYLUM ANIMALS
DNA SEQUENCE
RADIO ASTRONOMY
ECOLOGY MOVEMENT
NATURAL ECOLOGY
SURFACE TEMPERATURE
SURFACE WATER
STOCK PRICE
STOCK MARKET
INCOME TAX
SHIP TRANSPORT
EMPEROR CHARLES
KING CHARLES
CHARLES DE
EMPEROR NAPOLEON
SKIN CANCER
HUMAN SKIN
SUN MOON
SYSTEM SUN
EXTINCT SPECIES
MEDICAL SCIENCE
INTERNATIONAL STANDARD
STANDARD MODEL
STATISTICAL ANALYSIS
MONEY MARKET
QUANTUM FIELD
QUANTUM THEORY
QUANTUM STATES
PRIME MINISTER
PRIME NUMBER
BRITISH PRIME
TELEVISION NETWORK
COMPUTER NETWORK
NETWORK HTTP
SOLAR POWER
SOLAR SYSTEM
KJ MOL
HEAT ENGINE
PHYSICAL UNIVERSE
NUCLEAR MAGNETIC
MAGNETIC FIELD
COMPUTER SECURITY
NATIONAL SECURITY
NORTH AMERICAN
NORTH AMERICA
NORTH AFRICA
NORTH SEA
BRITISH NORTH
RADIO SIGNAL
WAVE EQUATION
SPANISH CIVIL
SPANISH WORD
SPANISH LANGUAGE
LAKE CITY
ROCK BAND
MILLION YEARS
MILLION LIGHT
MATERIAL USED
NAME USED
SYSTEM USED
PATENT LAW
IONIZATION ENERGY
ENGLISH WORD
ENGLISH NAME
ENGLISH LANGUAGE
ENGLISH FRENCH
INTERNATIONAL UNION
SOVIET UNION
ECONOMIC ANALYSIS
UNIVERSITY LIBRARY
LIBRARY HTTP
ARAB REPUBLIC
UNITED ARAB
ARAB STATES
AMERICAN COLONIES
BRITISH COLONIES
FRENCH COLONIES
CANCER CELLS
CHINESE LANGUAGE
YEARS WAR
CHRISTIAN BELIEF
COLLEGE LONDON
SOFTWARE ARCHITECTURE
SOFTWARE ENGINEERING
SOFTWARE DEVELOPMENT
SOFTWARE DESIGN
COMPUTER SOFTWARE
GAS LIQUID
LIQUID WATER
RADIO FREQUENCY
DATA STRUCTURE
ATOMIC STRUCTURE
ACTOR MUSICIAN
SOIL WATER
AGRICULTURAL PRODUCTION
GREEN MOVEMENT
HUMAN RACE

DNA MOLECULES
CHEMICAL BOND
PLANET EARTH
COMPUTER ARCHITECTURE
HUMAN EVOLUTION
PHYSICAL CHEMISTRY
OXIDATION NUMBER
CHEMICAL INDUSTRY
RADIO TELEVISION
TELEVISION PROGRAM
MARKET PRICE
NATURAL PHILOSOPHY
NATURAL CAPITAL
NATURAL NUMBER
NATURAL LAW
NATURAL SCIENCE
GREEN PARTY
NUCLEAR POWER
NUCLEAR TECHNOLOGY
NUCLEAR FORCE
NUCLEAR ENERGY
NUCLEAR WAR
GAS EQUATION
UNITED KINGDOM
ELECTRIC FIELD
FIELD THEORY
FRUIT TREE
CHRISTIAN CHURCH
CHRISTIAN RELIGION
BLOOD CELLS
HUMAN BLOOD
ROMAN CALENDAR
POPE JOHN
RIGHT MATH
CARBON COMPOUNDS
ROMAN EMPEROR
GERMAN EMPEROR
NOBEL PRIZE
INTERNATIONAL TRADE
INTERNATIONAL LAW
INTERNATIONAL SYSTEM
PRESIDENT JOHN
GENETIC ENGINEERING
COMPUTER ENGINEERING
SCIENCE ENGINEERING
ROCK MUSIC
ROCK GROUP
FAMILY TREE
GERMANY FRANCE
FRANCE GERMANY
PROTEIN MOLECULES
SOCIAL DEVELOPMENT
ECONOMIC DEVELOPMENT
HUMAN DEVELOPMENT
DEVELOPMENT THEORY
WWW NOBEL
CHEMICAL COMPOUNDS
AMERICAN CIVIL
AMERICAN SINGER
AMERICAN FILM
AMERICAN ACTRESS
AMERICAN CULTURE
AMERICAN PRESIDENT
AMERICAN ACTOR
AMERICAN WAR
AMERICAN STATES
COMPUTER GAME
GENETIC MATERIAL
CIVIL LAW
CIVIL WAR
GREEK MYTHOLOGY
ROMAN MYTHOLOGY
BUSINESS MODEL
CHEMICAL REACTION
MANAGEMENT TOPICS
GREEK WORD
GREEK GOD
GREEK PHILOSOPHY
GREEK LANGUAGE
SOVIET GOVERNMENT
WAR BATTLE
STUDY MEDICINE
MEDICINE HTTP
MANAGEMENT SYSTEM
TIME PERIOD
PHILOSOPHICAL SYSTEM
EARTH METAL
EARTH ORBIT
EARTH SCIENCE
MODEL ORGANISMS
ELECTRIC POWER
ELECTRIC LIGHT
MUSIC SINGER
SINGER ACTRESS
ACTRESS SINGER
SINGER ACTOR
ACTOR SINGER
SPACE PROGRAM
ATOMIC MASS
HEAT ENERGY
POLITICAL POWER
TRANSPORT SYSTEM
POLITICAL PARTY
INFORMATION TECHNOLOGY
CENTURY BC
UNIVERSITY COLLEGE
ART CULTURE
INFORMATION SCIENCE
INFORMATION HTTP
INFORMATION SYSTEM
INFORMATION THEORY
WATER MOLECULES
FRENCH KING
ROMAN EMPIRE
GERMAN EMPIRE
BRITISH EMPIRE
FRENCH EMPIRE
SOCIAL CLASS
SOCIAL CAPITAL
SOCIAL LIFE
SOCIAL POLITICAL
SOCIAL SCIENCE
SOCIAL THEORY
SOCIETY RELIGION
HUMAN SOCIETY
ROMAN REPUBLIC
FRENCH REPUBLIC
COMPUTER PROGRAM
ROMAN PERIOD
GROUP PERIOD
SEA WATER
MARKET ECONOMY
POLITICAL ECONOMY
LAW SCHOOL
GERMAN NATIONAL
GERMAN LANGUAGE
FRENCH GERMAN
GERMAN STATES
ECONOMIC COMMUNITY
POLITICAL HISTORY
HISTORY HTTP
STATES HISTORY
MARKET VALUE
ECONOMIC VALUE
VALUE THEORY
FILM ACTRESS
NATIONAL FILM
FILM PRODUCTION
FRENCH FILM
FILM ACTOR
PHYSICAL OBJECTS
PHYSICAL LAW
PHYSICAL SCIENCE
PHYSICAL SYSTEM
PHYSICAL THEORY
MARKET SYSTEM
POLITICAL PHILOSOPHY
SCIENCE PHILOSOPHY
PHILOSOPHY HTTP
CHEMICAL ENERGY
ATOMIC ENERGY
ENERGY HTTP
ENERGY STATES
CHEMICAL ELEMENTS
FOOD PRODUCTION
FOOD WATER
BRITISH GOVERNMENT
NATIONAL GOVERNMENT
FRENCH GOVERNMENT
GOVERNMENT HTTP
STATES GOVERNMENT
HUMAN CAPITAL
CAPITAL CITY
ECONOMIC POLITICAL
ECONOMIC SYSTEM
ECONOMIC THEORY
POLITICAL LIFE
FAMILY LIFE
HUMAN LIFE
LIFE HTTP
FEBRUARY MARCH
ATOMIC NUMBER
NUMBER SYSTEM

NUMBER THEORY
BRITISH ACTOR
WAR BRITISH
HUMAN POPULATION
COMPUTER SCIENCE
COMPUTER SYSTEM
LAW SYSTEM
LAW STATES
JANUARY FEBRUARY
DE FRANCE
HTTP DE
PLANTS ANIMALS
NATIONAL DAY
STAR SYSTEM
NATIONAL LANGUAGE
NATIONAL SCIENCE
FRENCH NATIONAL
STATES NATIONAL
LANGUAGE CULTURE
HUMAN CULTURE
POLITICAL SCIENCE
POLITICAL SYSTEM
POLITICAL THEORY
FAMILY NAME
LANGUAGE FAMILY
HUMAN LANGUAGE
FRENCH LANGUAGE
HTTP WWW
GROUP HTTP
FRANCE HTTP
FRENCH PRESIDENT
STATES PRESIDENT
HUMAN SPECIES
SCIENCE HTTP
CENTURY FRENCH
UNIVERSITY HTTP
CITY STATES
SYSTEM HTTP
STATES HTTP
UNITED STATES

Appendix H

Sample Keyword Entries

Entry for doc 2: Agriculture - Wikipedia.
Keywords: AGRICULTURAL (0.50) FARMING (0.34) FOOD (0.29) CROP (0.26) ANIMALS (0.21) PLANTS (0.19) AGRICULTURAL_POLICY (0.18) LOCAL_FOOD (0.16) GENETIC (0.14) PRODUCTION (0.13)

Entry for doc 3: Anthropology - Wikipedia.
Keywords: ANTHROPOLOGY (0.38) SOCIAL (0.31) PHYSICAL_ANTHROPOLOGY (0.27) LINGUISTICS (0.23) CULTURE (0.22) BASIC_TOPICS (0.20) STUDY (0.18) HUMAN (0.16) ANTHROPOLOGISTS (0.11) PHYSICAL (0.11)

Entry for doc 4: Archaeology - Wikipedia.
Keywords: ARCHAEOLOGY (1.09) ARCHAEOLOGISTS (0.26) EXCAVATIONS (0.17) SITE (0.14) CULTURE (0.11) RECONSTRUCTION (0.08) STUDY (0.07) ANALYSIS (0.07) METHOD (0.06) LEWIS (0.06)

Entry for doc 6: Astronomy and astrophysics - Wikipedia.
Keywords: ASTRONOMY (0.45) INFRARED (0.12) RADIO_ASTRONOMY (0.12) ASTRONOMICAL (0.11) OPTICAL (0.11) OBSERVATORY (0.10) TELESCOPE (0.10) OBSERVED (0.08) INTERSTELLAR_MEDIUM (0.08) UV (0.08) STELLAR (0.08) WAVELENGTH (0.08)

Entry for doc 7: Biology - Wikipedia.
Keywords: BIOLOGY (0.31) DEVELOPMENTAL_BIOLOGY (0.30) BASIC_TOPICS (0.15) ORGANISMS (0.15) MODEL_ORGANISMS (0.12) DEVELOPMENTAL (0.12) MOLECULAR (0.12) EVOLUTIONARY (0.12) PHYLOGENY (0.10) ONTOGENY (0.10) LIFE (0.09)

Entry for doc 8: Business - Wikipedia.
Keywords: BUSINESS (0.28) INDUSTRY (0.21) TOPICS (0.19) CORPORATION (0.17) ECONOMIC (0.17) LIST (0.16) SECTOR (0.11) INVESTMENT (0.09) ELECTRON (0.09) ACCOUNT (0.09)

Entry for doc 9: Chemistry - Wikipedia.
Keywords: CHEMISTRY (0.59) CHEMICAL (0.29) IONIZATION_ENERGY (0.22) COMPOUNDS (0.19) ACID (0.17) BASIC_TOPICS (0.15) REACTION (0.15) CHEMICAL_EQUILIBRIUM (0.12) PHYSICAL_CHEMISTRY (0.12) ELEMENTS (0.11)

Entry for doc 10: Communication - Wikipedia.
Keywords: COMMUNICATION (0.58) BASIC_TOPICS (0.20) NETWORK (0.17) TECHNOLOGY (0.16) MEDIA (0.11) INFORMATION (0.10) PEOPLE (0.10) LANGUAGE (0.09) FIELD (0.09) TOPICS (0.08) TELEPHONE (0.08)

Entry for doc 11: Computer science - Wikipedia.
Keywords: COMPUTER (0.73) SOFTWARE_ENGINEERING (0.38) SCIENCE (0.25) ENGINEERING (0.20) SOFTWARE (0.17) INFORMATION (0.12) MATHEMATICS (0.12) PATTERN (0.10) ALGORITHM (0.10) THEORY (0.09)

Entry for doc 12: Earth science - Wikipedia.
Keywords: SCIENCE (0.30) GEOGRAPHY (0.22) EARTH (0.14) ENVIRONMENTAL

(0.13) GEOLOGY (0.13) EARTH_SCIENCE (0.12) ROCK (0.10) STUDY (0.09) MAP
(0.07) PHYSICAL (0.07)

Appendix I

Sample More-Like-This Entries

Entry for doc 2: Agriculture - Wikipedia.

- Agriculture - Wikipedia. (1000.00)
- Collective farming - Wikipedia. (999.21)
- Agricultural policy - Wikipedia. (999.00)
- Agricultural policy - Wikipedia. (998.99)
- Agricultural productivity - Wikipedia. (998.98)
- Monoculture - Wikipedia. (998.97)
- Local food - Wikipedia. (998.39)
- Local food - Wikipedia. (998.39)
- Genetically modified food - Wikipedia. (994.79)
- Agribusiness - Wikipedia. (989.58)

Entry for doc 3: Anthropology - Wikipedia.

- Cultural anthropology - Wikipedia. (1000.00)
- Ethnography - Wikipedia. (999.91)
- Cultural anthropology - Wikipedia. (999.79)
- Diffusion (anthropology) - Wikipedia. (997.22)
- Cultural evolution - Wikipedia. (997.06)
- Anthropology of religion - Wikipedia. (995.39)
- Medical anthropology - Wikipedia. (990.00)
- Claude LÃ©vi-Strauss - Wikipedia. (989.45)
- Physical anthropology - Wikipedia. (980.00)
- Linguistic anthropology - Wikipedia. (977.61)

Entry for doc 4: Archaeology - Wikipedia.

- Archaeology - Wikipedia. (1000.00)
- Archaeology - Wikipedia. (1000.00)
- Post-processualism - Wikipedia. (999.92)
- Lewis Binford - Wikipedia. (999.82)
- Processualism - Wikipedia. (999.70)
- Systems theory in Archaeology - Wikipedia. (997.71)
- Nautical archaeology - Wikipedia. (990.65)
- James Deetz - Wikipedia. (981.31)
- Reconstruction archaeology - Wikipedia. (971.96)
- Archaeological site - Wikipedia. (971.93)

Entry for doc 6: Astronomy and astrophysics - Wikipedia.

- Astronomy and astrophysics - Wikipedia. (1000.00)
- Astronomy and astrophysics - Wikipedia. (1000.00)
- Infrared astronomy - Wikipedia. (994.60)
- UKIRT - Wikipedia. (992.62)
- David Fabricius - Wikipedia. (987.18)
- Amateur astronomy - Wikipedia. (986.80)
- Kitt Peak National Observatory - Wikipedia. (984.83)
- Space observatory - Wikipedia. (984.32)
- Lowell Observatory - Wikipedia. (983.82)
- Clyde Tombaugh - Wikipedia. (982.89)

Entry for doc 7: Biology - Wikipedia.

- Biology - Wikipedia. (1000.00)
- Embryology - Wikipedia. (999.55)
- Ontogeny - Wikipedia. (999.54)
- Phenetics - Wikipedia. (999.47)
- Cell biology - Wikipedia. (998.89)
- Evolutionary developmental biology - Wikipedia. (979.66)
- Developmental biology - Wikipedia. (979.66)
- Xenopus laevis - Wikipedia. (979.49)
- Phylogeny - Wikipedia. (935.59)
- Teratology - Wikipedia. (928.81)

Entry for doc 8: Business - Wikipedia.

- Business - Wikipedia. (1000.00)
- Asset - Wikipedia. (996.42)
- Business plan - Wikipedia. (993.41)
- Horizontal integration - Wikipedia. (992.52)
- Investment - Wikipedia. (986.30)
- Ecological model of competition - Wikipedia. (980.80)
- Venture capital - Wikipedia. (972.60)
- Annuity - Wikipedia. (972.44)
- Marketing myopia - Wikipedia. (958.90)
- Barter - Wikipedia. (945.21)

Entry for doc 9: Chemistry - Wikipedia.

- Nobel Prize in Chemistry - Wikipedia. (1000.00)
- Nobel Prize in Chemistry - Wikipedia. (1000.00)
- Antoine Lavoisier - Wikipedia. (998.51)
- Chemical synthesis - Wikipedia. (998.20)
- Antoine Jerome Balard - Wikipedia. (983.33)
- Organic compound - Wikipedia. (979.27)
- Biochemistry - Wikipedia. (978.08)
- Putrescine - Wikipedia. (976.89)
- Nuclear chemistry - Wikipedia. (966.67)
- Ferrocene - Wikipedia. (962.43)

Entry for doc 10: Communication - Wikipedia.

- Submarine communications cable - Wikipedia. (1000.00)
- Smartmob - Wikipedia. (997.96)
- Wireless network - Wikipedia. (995.64)
- Wireless network - Wikipedia. (995.59)
- Mobile phone - Wikipedia. (994.01)
- Metcalfe's law - Wikipedia. (991.71)
- World Wide Web - Wikipedia. (983.87)
- Internet - Wikipedia. (980.70)
- Persuasion technology - Wikipedia. (979.66)
- Spamming - Wikipedia. (978.49)

Entry for doc 11: Computer science - Wikipedia.

- Computer science - Wikipedia. (1000.00)
- Computer scientist - Wikipedia. (998.99)
- Ontology (computer science) - Wikipedia. (998.31)
- Interdisciplinarity - Wikipedia. (997.91)
- Frederick P. Brooks - Wikipedia. (982.30)
- Computer engineering - Wikipedia. (981.02)
- Computer programming - Wikipedia. (979.22)
- Computer programming - Wikipedia. (979.15)
- Atomic (computer science) - Wikipedia. (973.45)
- Foundation ontology - Wikipedia. (972.50)

Entry for doc 12: Earth science - Wikipedia.

- Earth science - Wikipedia. (1000.00)
- Geography - Wikipedia. (999.21)
- Geography - Wikipedia. (999.21)
- Geography - Wikipedia. (999.20)
- Hydrology - Wikipedia. (967.27)
- Geologist - Wikipedia. (927.27)
- Environmental geography - Wikipedia. (872.73)
- Geology - Wikipedia. (752.73)
- Environmental science - Wikipedia. (727.27)
- Regional science - Wikipedia. (727.20)

Appendix J

Performance Test

J.1 Performance Test Settings

J.1.1 Filtering

- Minimum document frequency for terms: 10
- Maximum document frequency for terms: 50 %
- Remove 1-character terms
- Remove numbers with less than 4 digits

J.1.2 Bigram Extraction

- Do not extract bigrams

J.1.3 Weighting

- Local weighting algorithm: Mondosoft's weighting scheme
- Global weighting algorithm: Log-Entropy

J.1.4 Keyword Extraction

- Keyword extraction algorithm: Synergy
- LSA dimension: 300
- Normalise document vectors
- Minimum keywords per page: 10
- Maximum keywords per page: 20
- Cut-off factor: 0.5

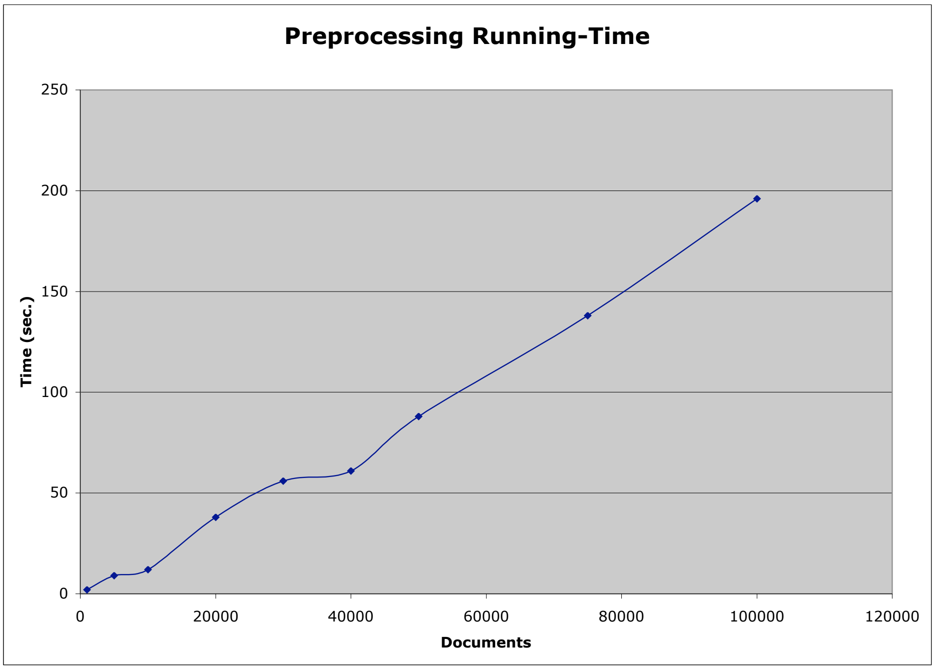
J.1.5 Clustering Algorithm Specific Settings

- Apriori minimum support: 10
- Hierarchy-based algorithm distance measure: Cosine

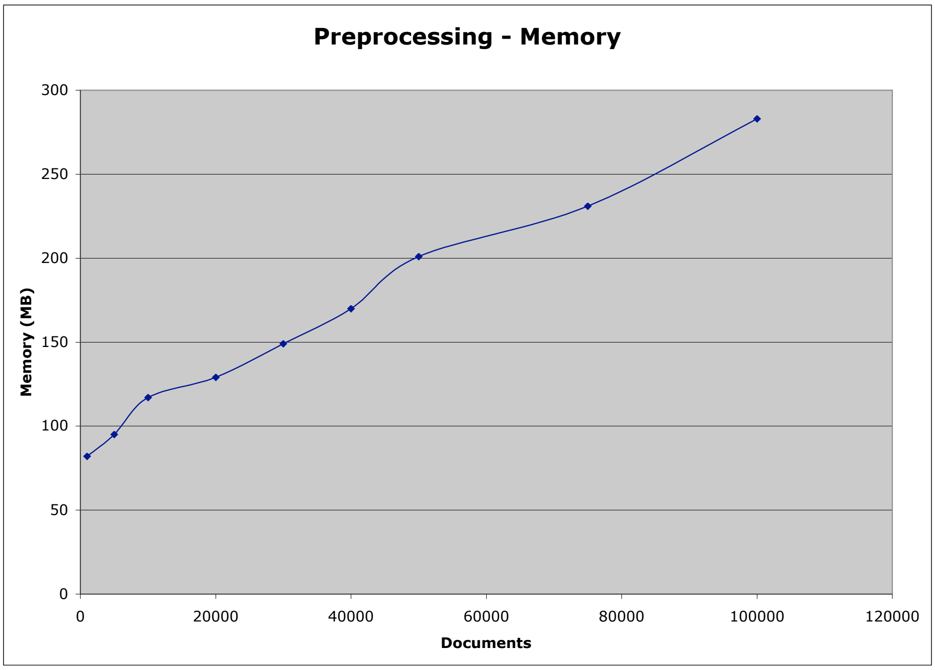
- Divisive algorithm maximum cluster size: 10
- K-Means maximum iterations per bisection: 500
- K-Means maximum error stop condition: 0.1 %
- CURE maximum representatives: 10
- CURE shrinking factor (α): 0.5

J.2 Preprocessing

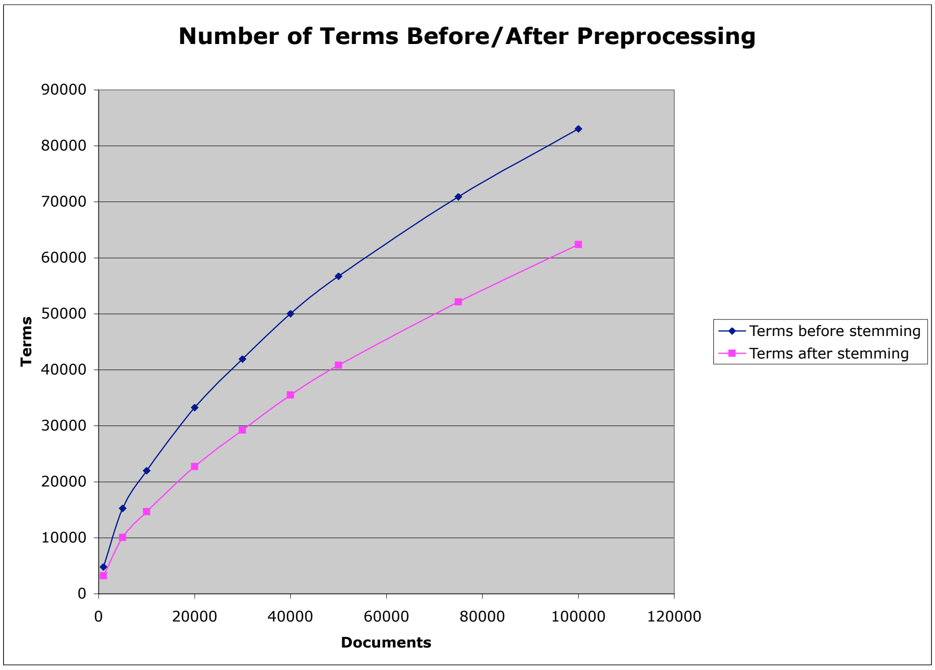
J.2.1 Preprocessing: Running-Time



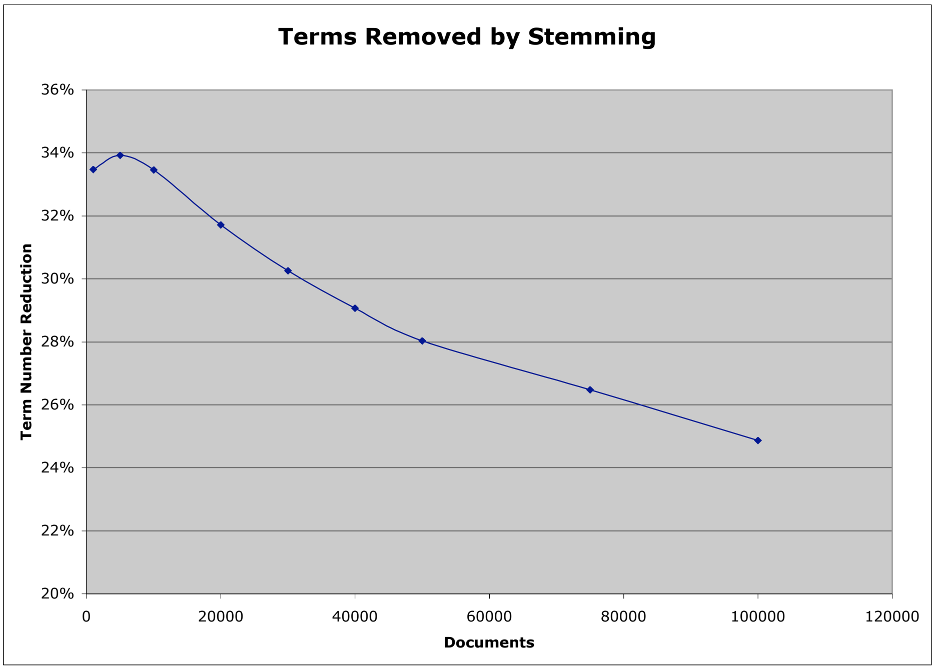
J.2.2 Preprocessing: Memory



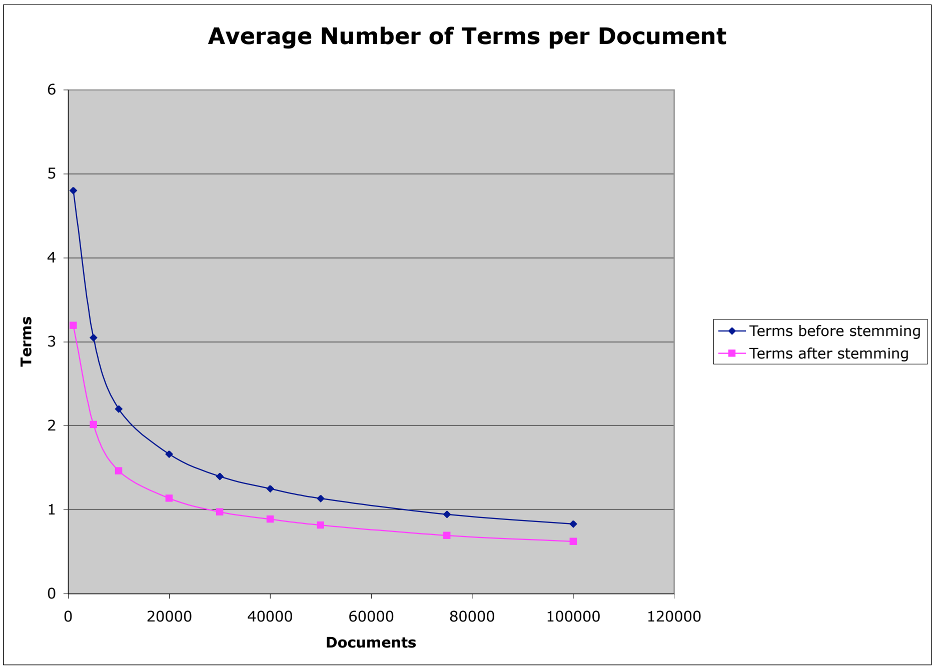
J.2.3 Preprocessing: Terms before/after Stemming



J.2.4 Preprocessing: Stemming Reduction

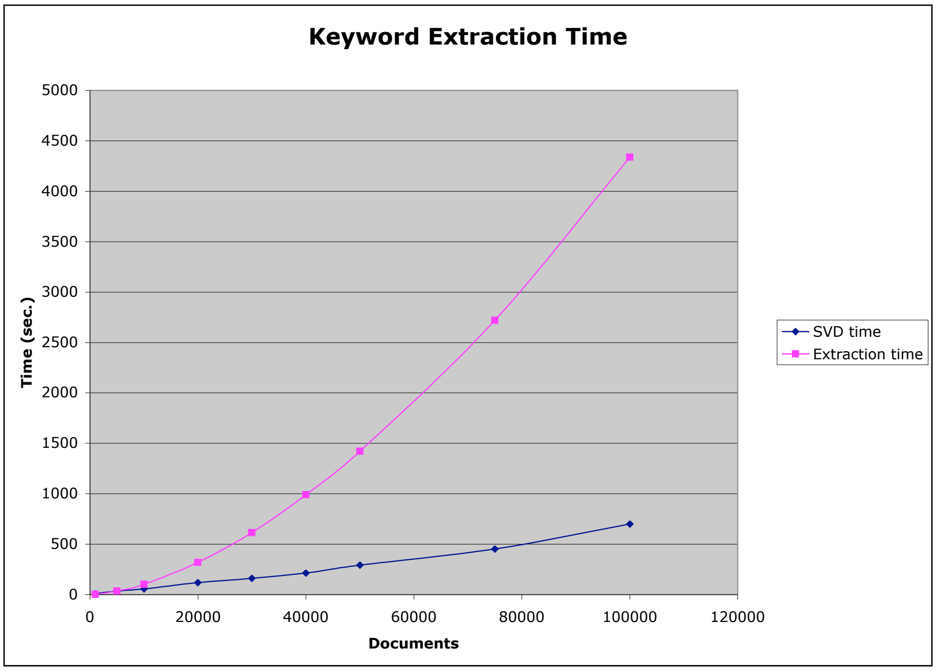


J.2.5 Preprocessing: Average New Term per Document

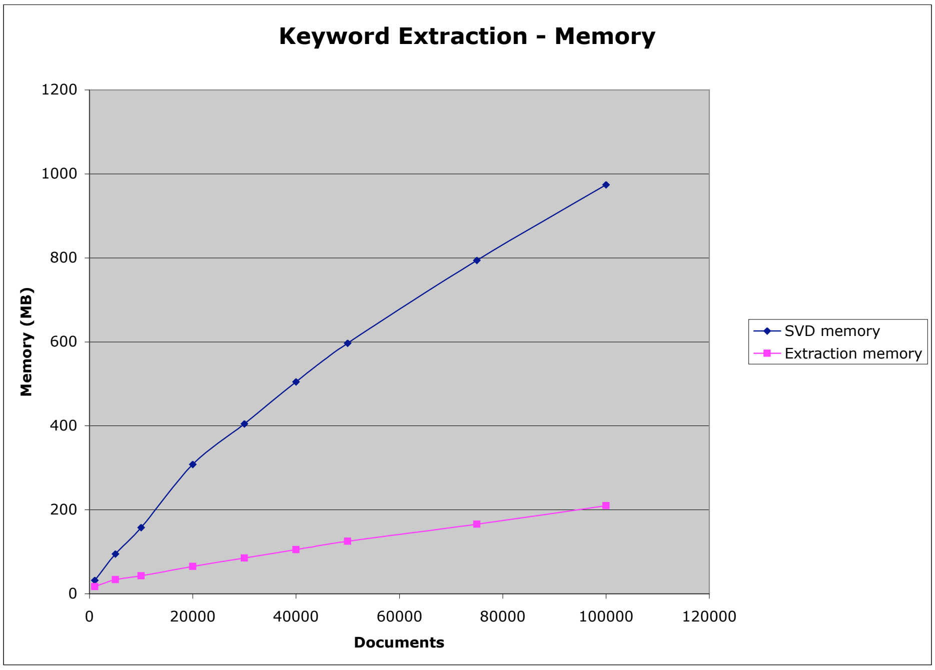


J.3 Keyword Extraction

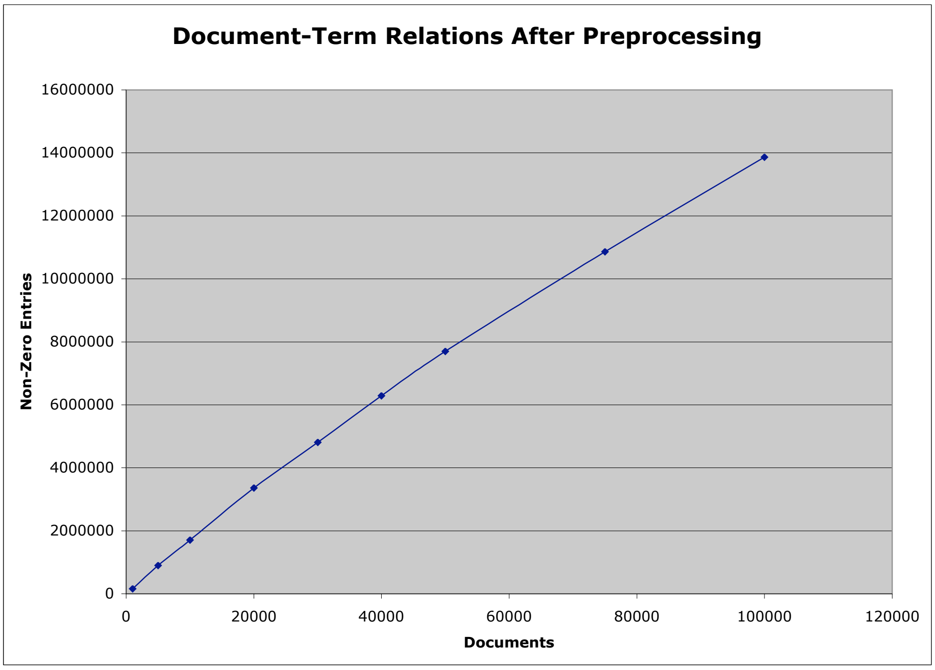
J.3.1 Keyword Extraction: Running-Time



J.3.2 Keyword Extraction: Memory

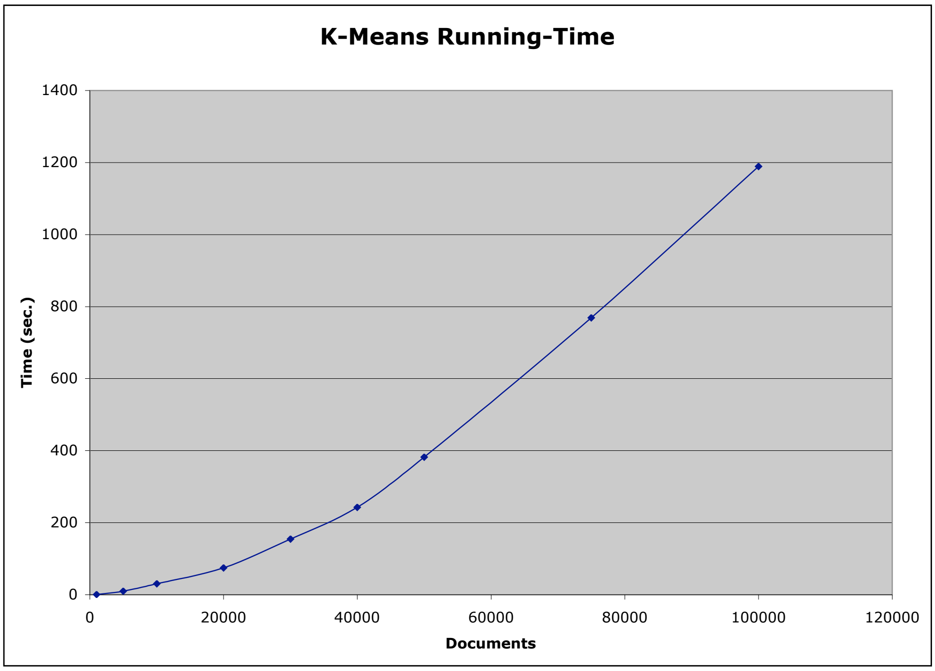


J.3.3 Keyword Extraction: Document-Term Relations

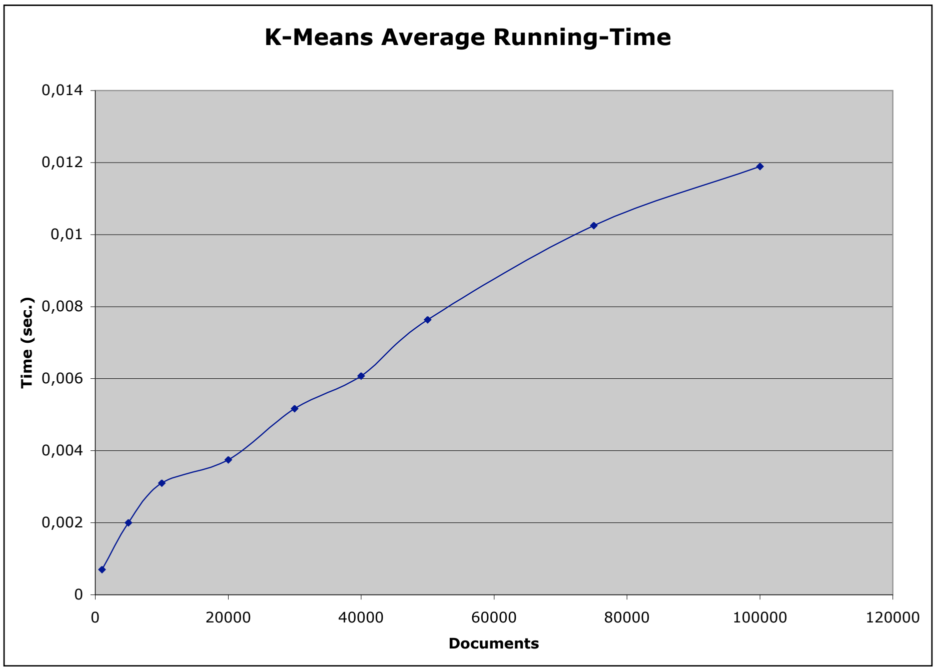


J.4 K-Means

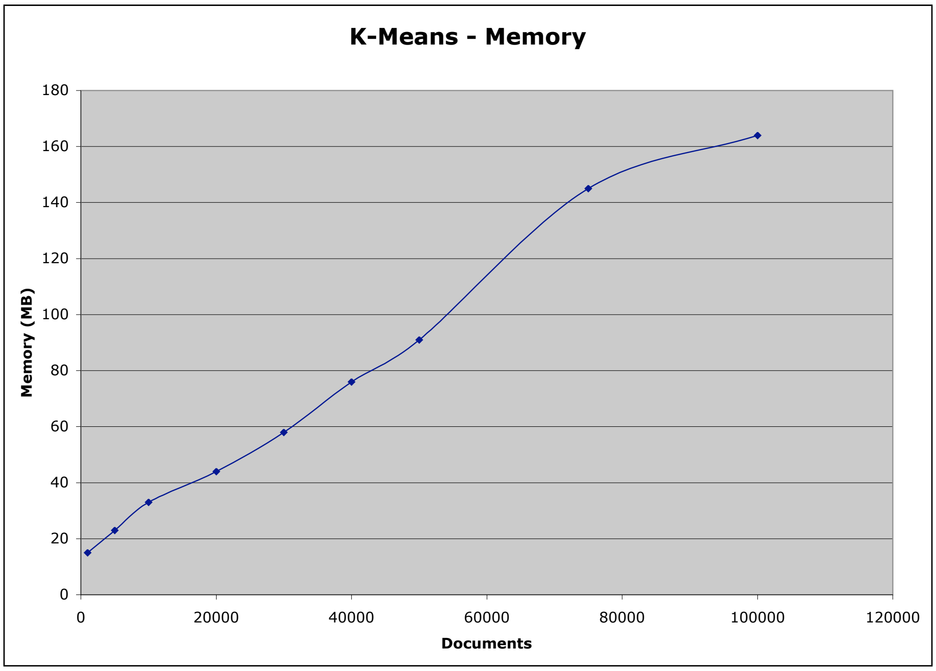
J.4.1 K-Means: Running-Time



J.4.2 K-Means: Average Running-Time

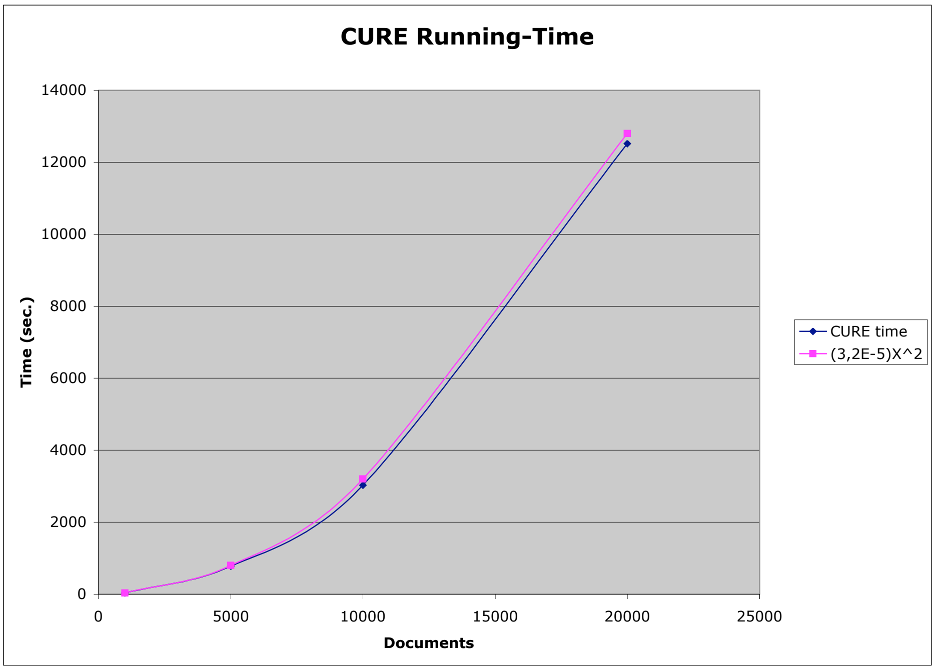


J.4.3 K-Means: Memory

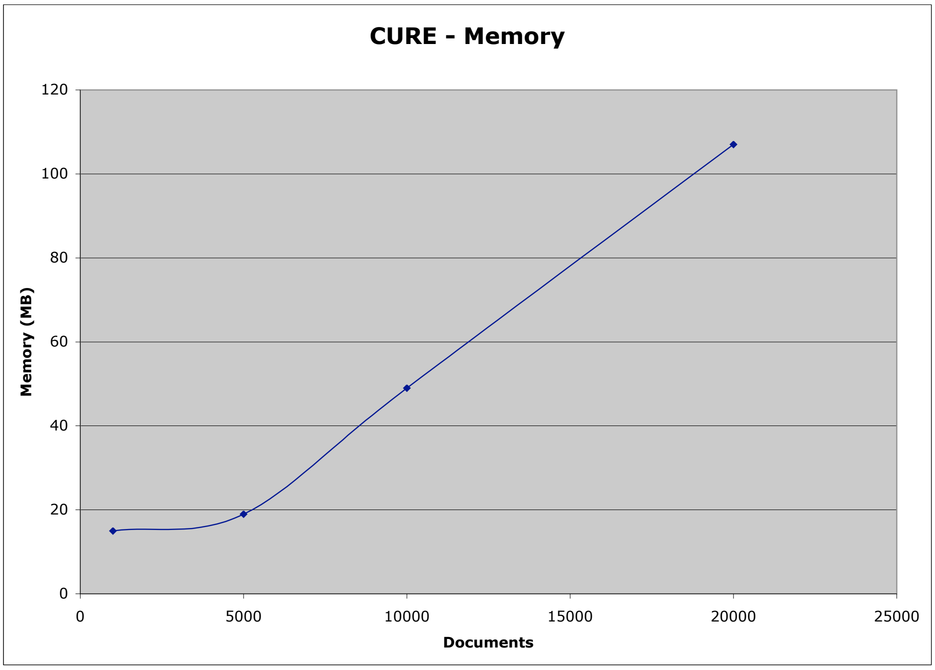


J.5 CURE

J.5.1 CURE: Running-Time

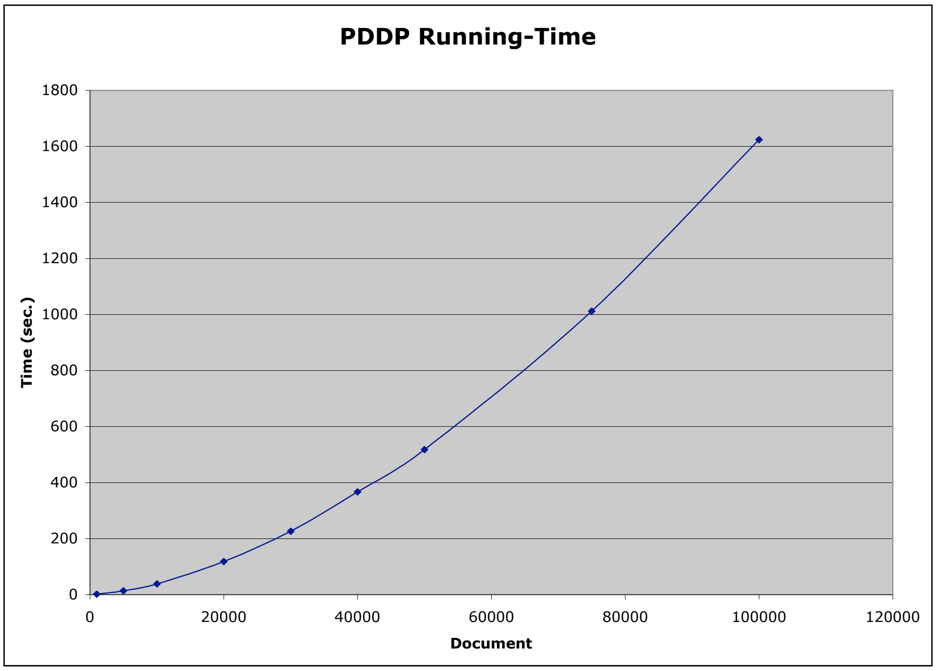


J.5.2 CURE: Memory

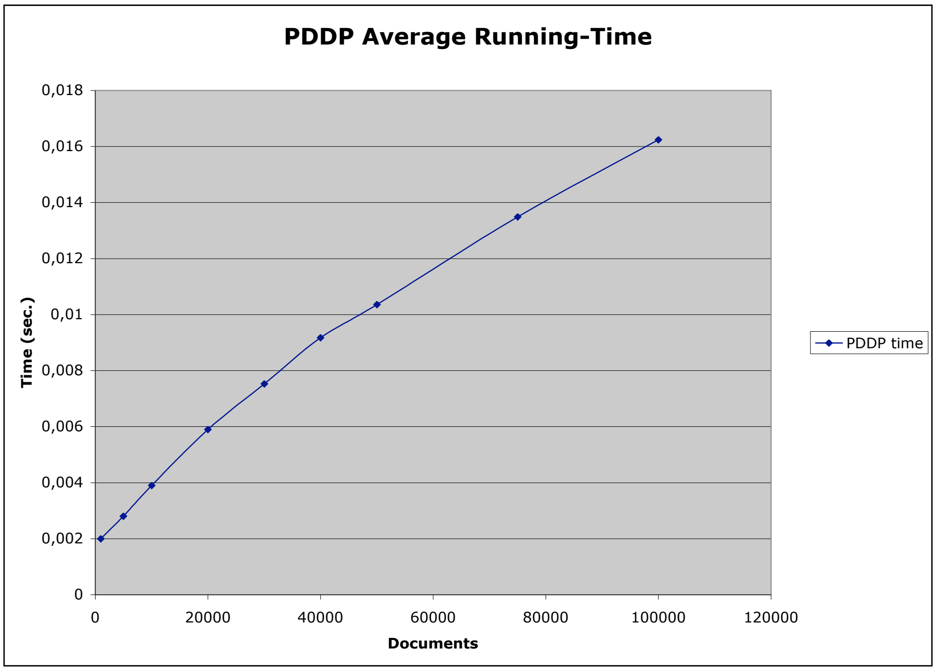


J.6 PDDP

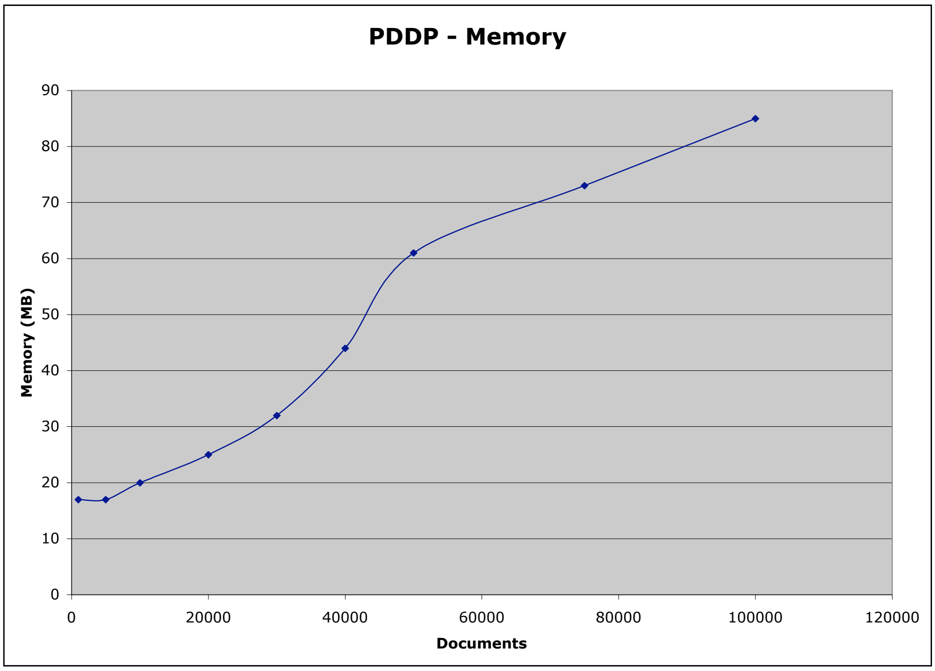
J.6.1 PDDP: Running-Time



J.6.2 PDDP: Average Running-Time

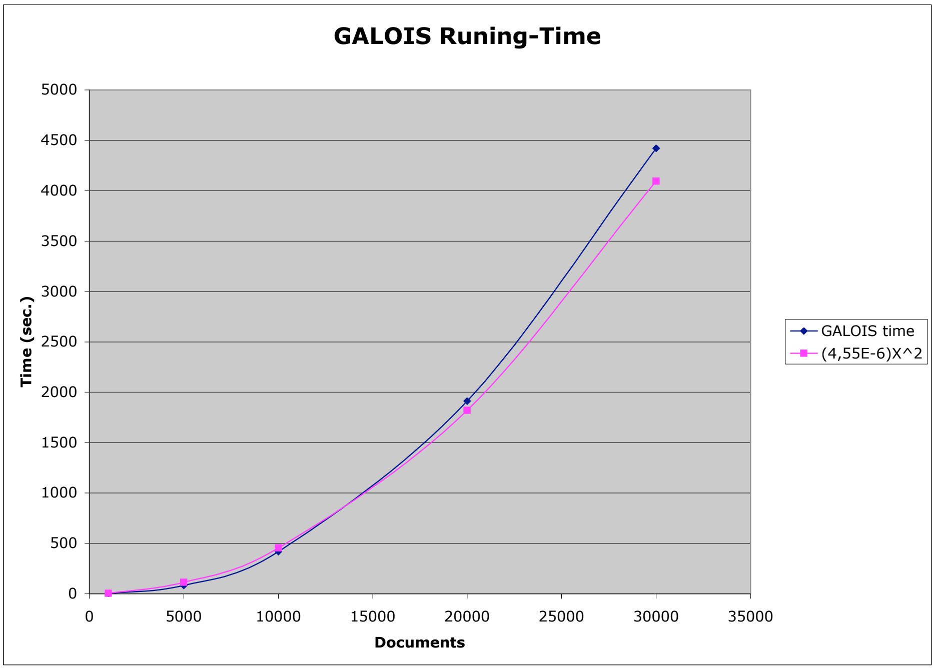


J.6.3 PDDP: Memory

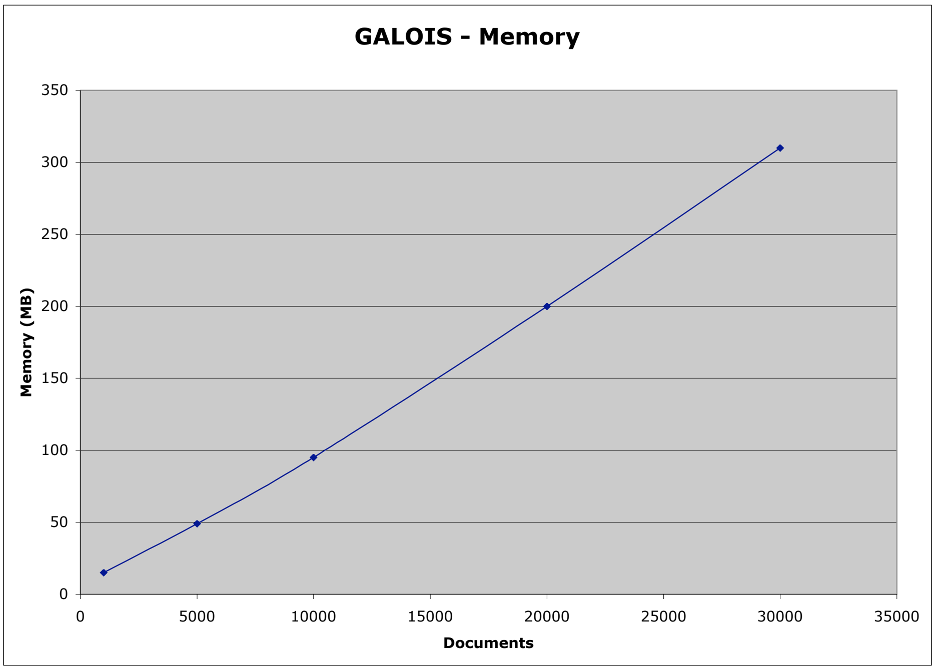


J.7 GALOIS

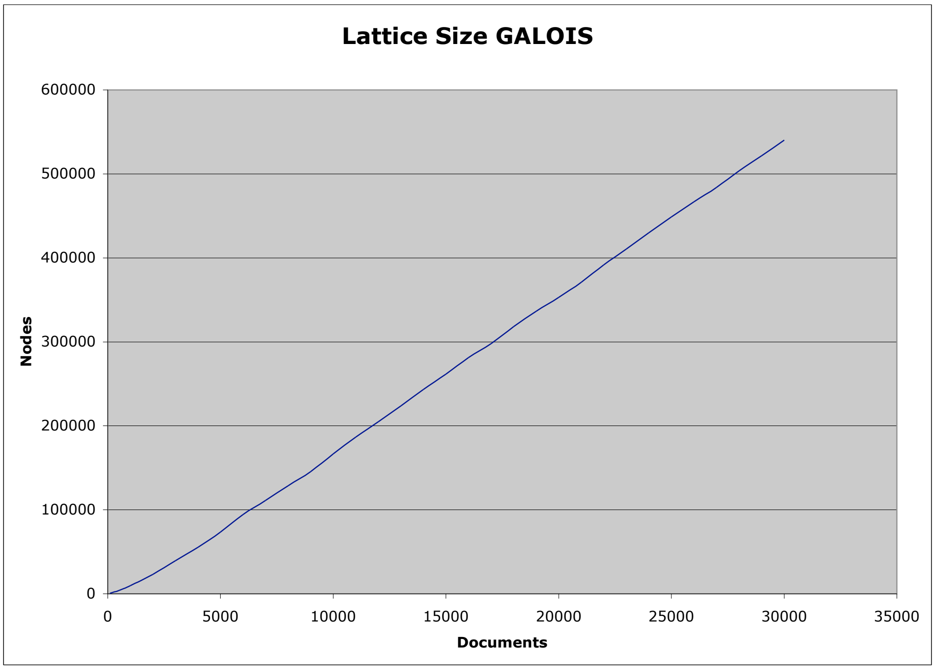
J.7.1 GALOIS: Running-Time



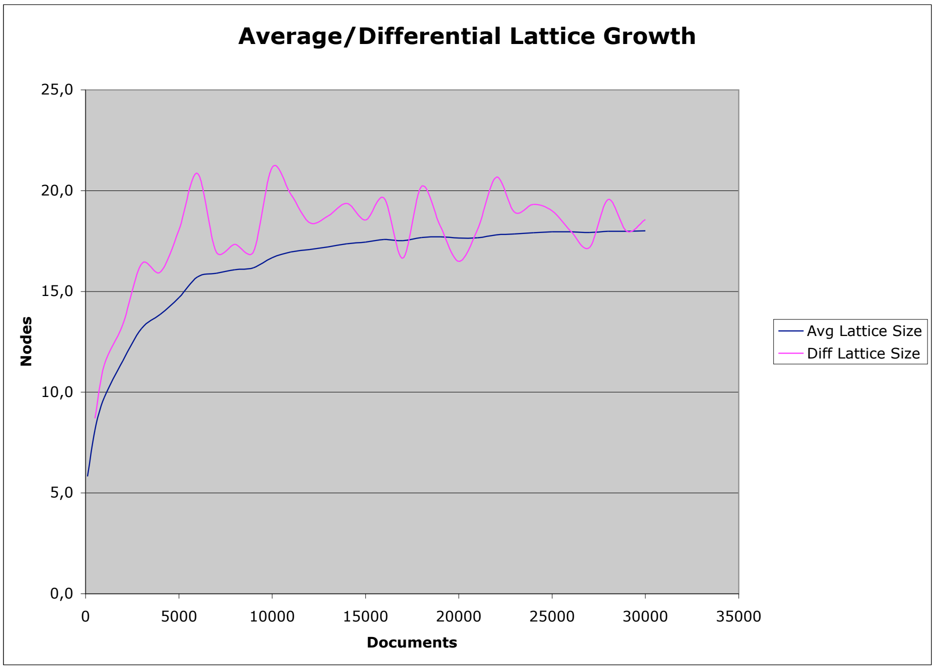
J.7.2 GALOIS: Memory



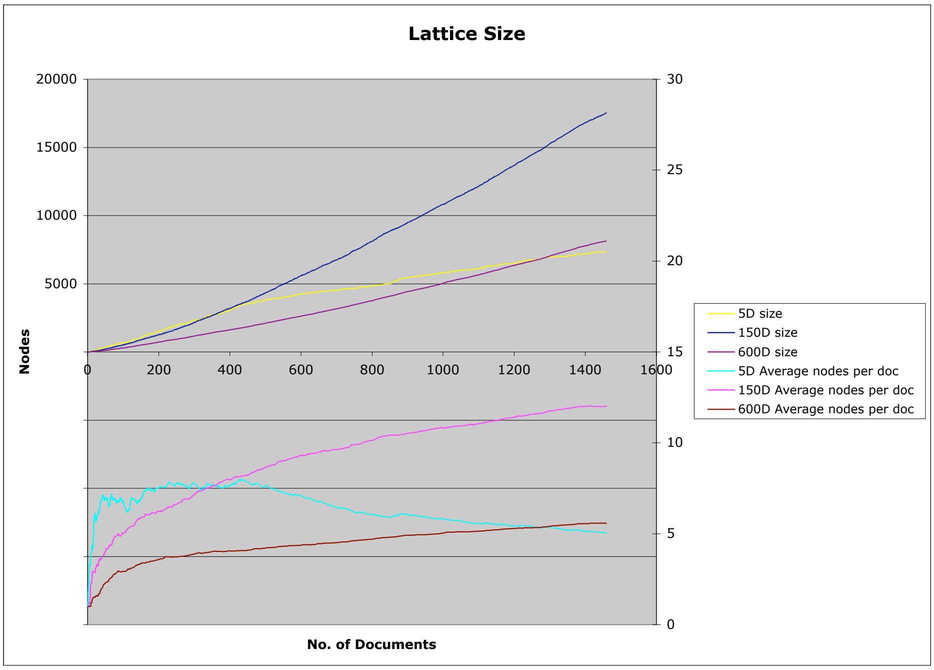
J.7.3 GALOIS: Lattice Size



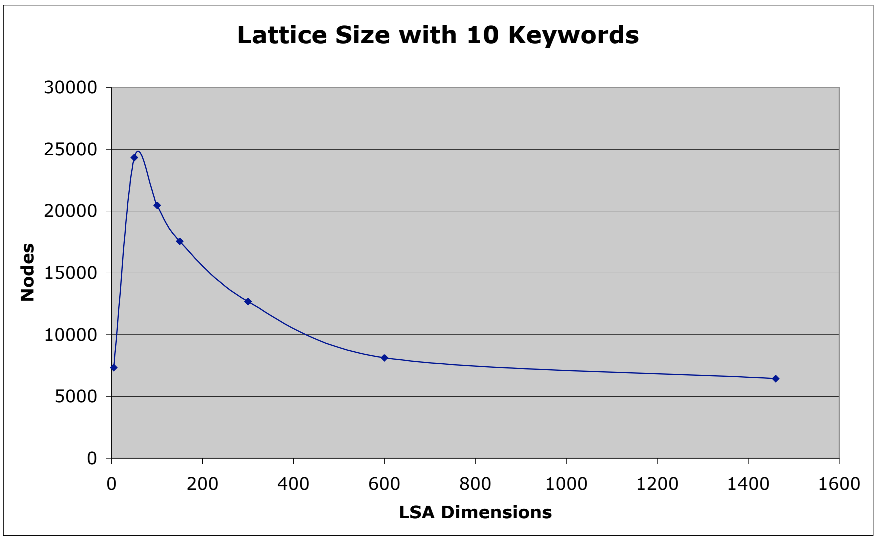
J.7.4 GALOIS: Lattice Growth



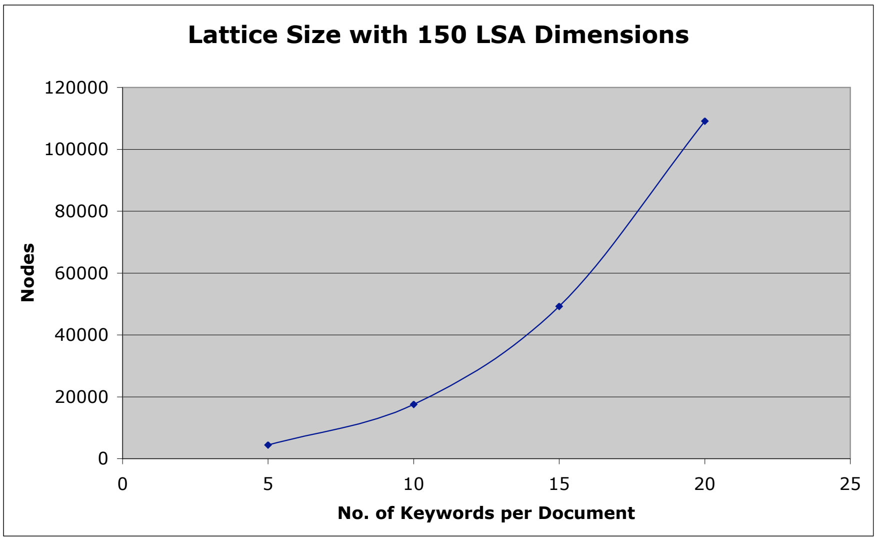
J.7.5 GALOIS: Lattice Size (CISI)



J.7.6 GALOIS: Lattice Size vs LSA Dimension (CISI)

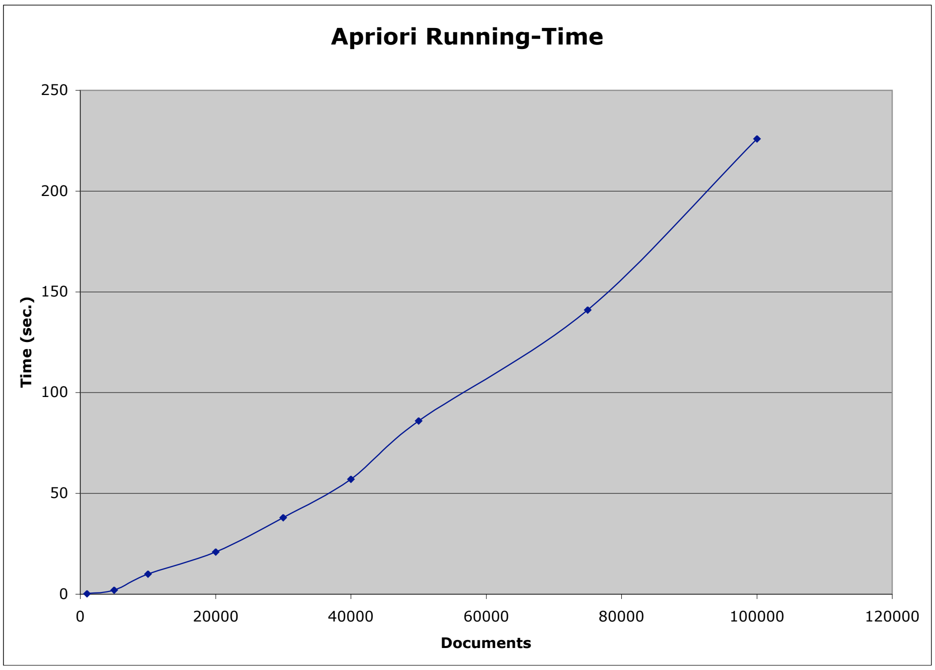


J.7.7 GALOIS: Lattice Size vs Number of Keywords (CISI)

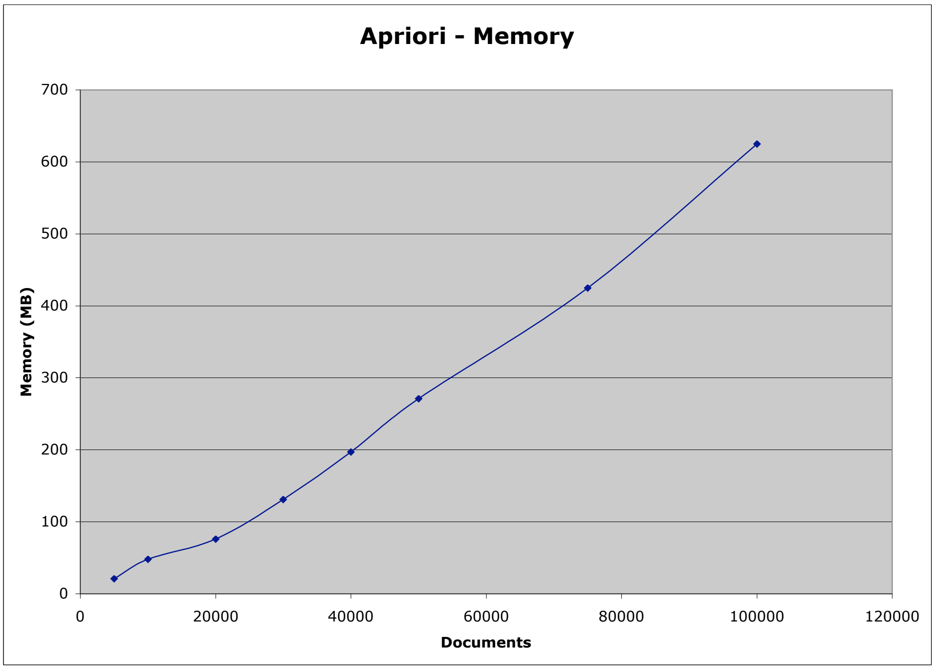


J.8 Apriori

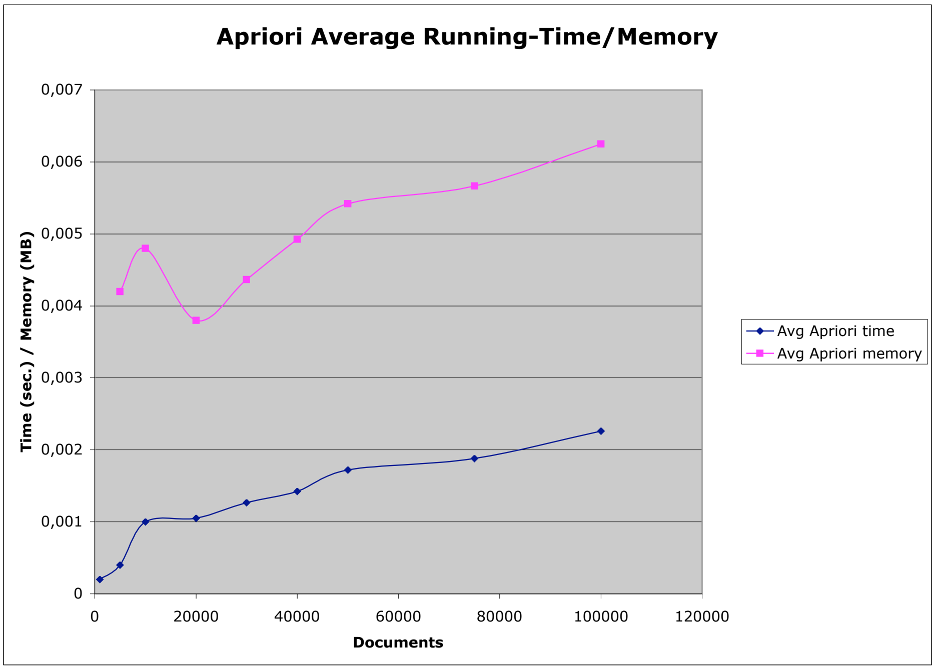
J.8.1 Apriori: Running-Time



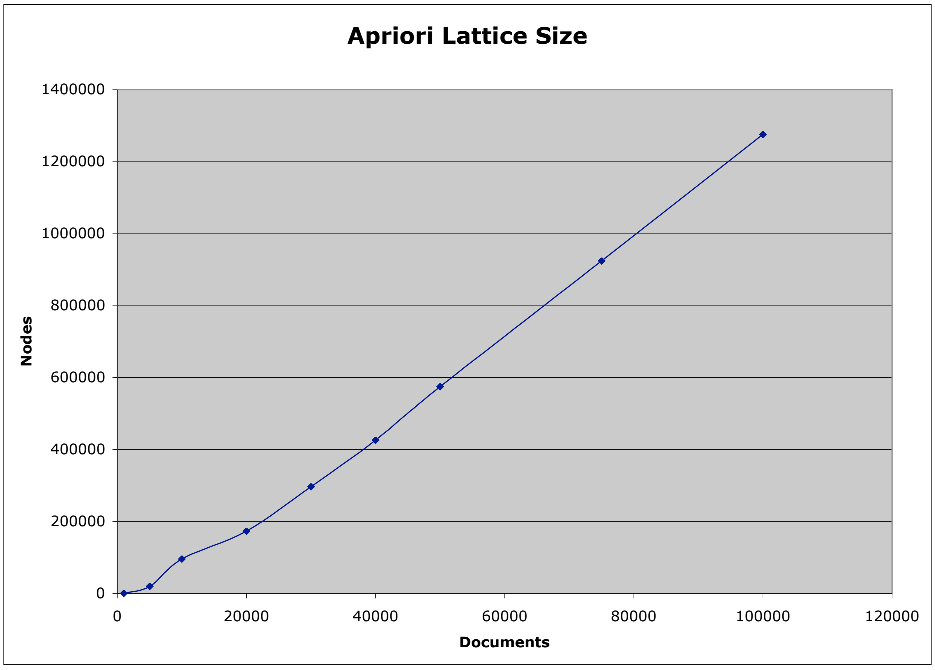
J.8.2 Apriori: Memory



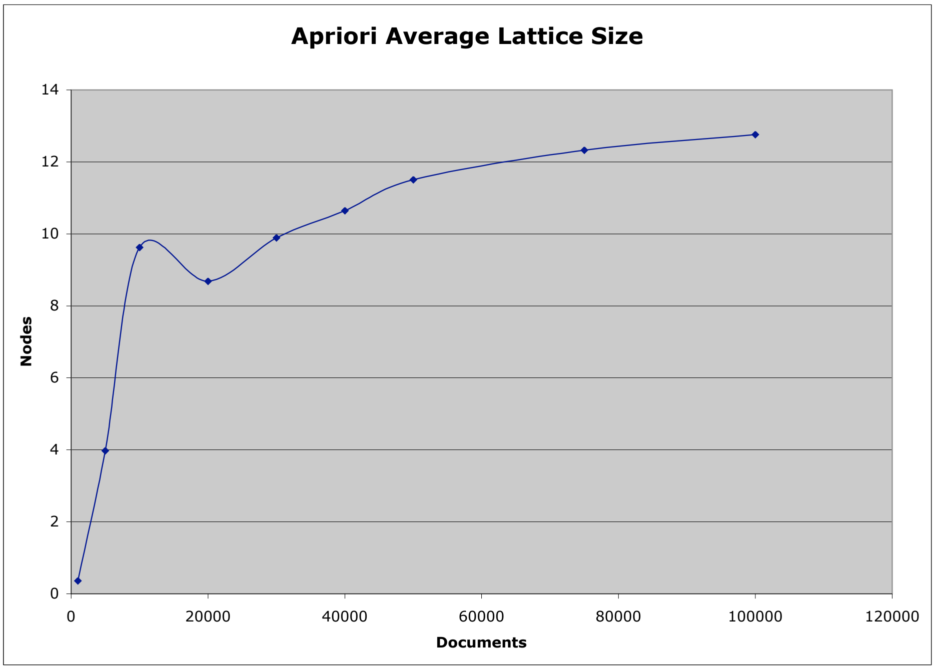
J.8.3 Apriori: Average Running-Time and Memory



J.8.4 Apriori: Lattice Size

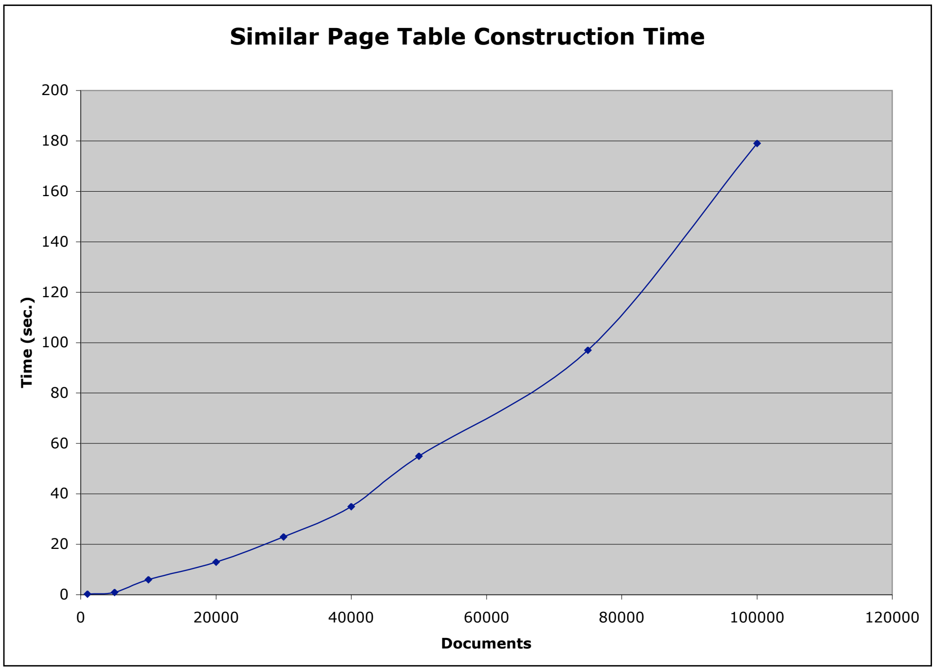


J.8.5 Apriori: Average Lattice Size

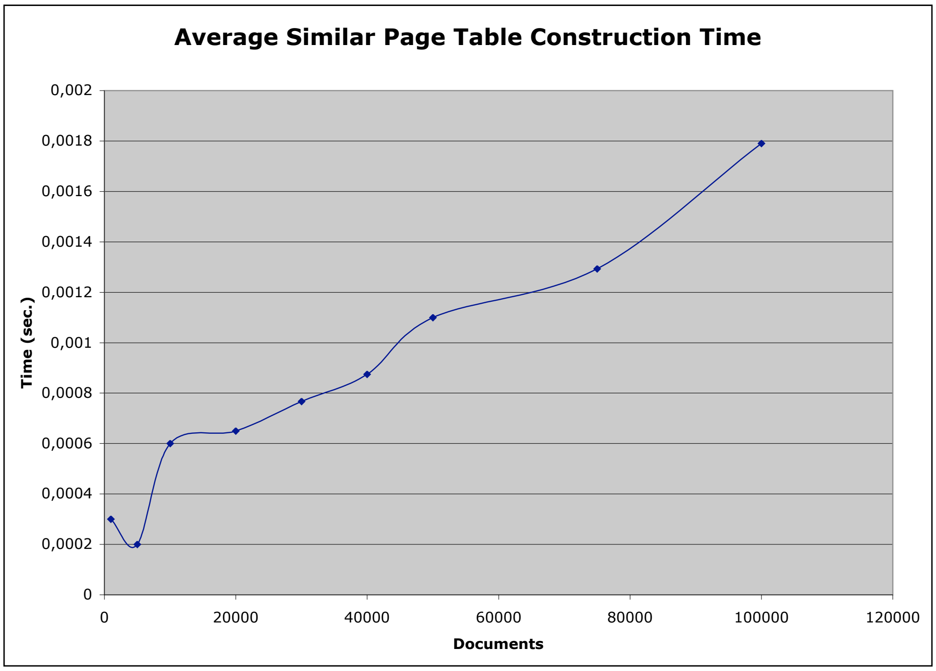


J.9 Similar Pages

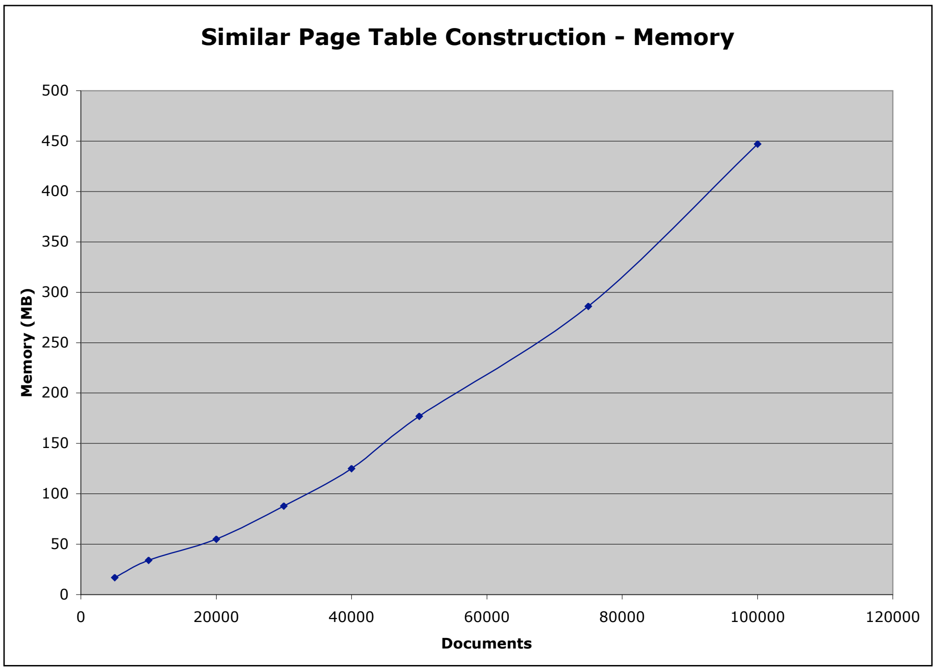
J.9.1 Similar Pages: Construction Time



J.9.2 Similar Pages: Average Construction Time



J.9.3 Similar Pages: Memory

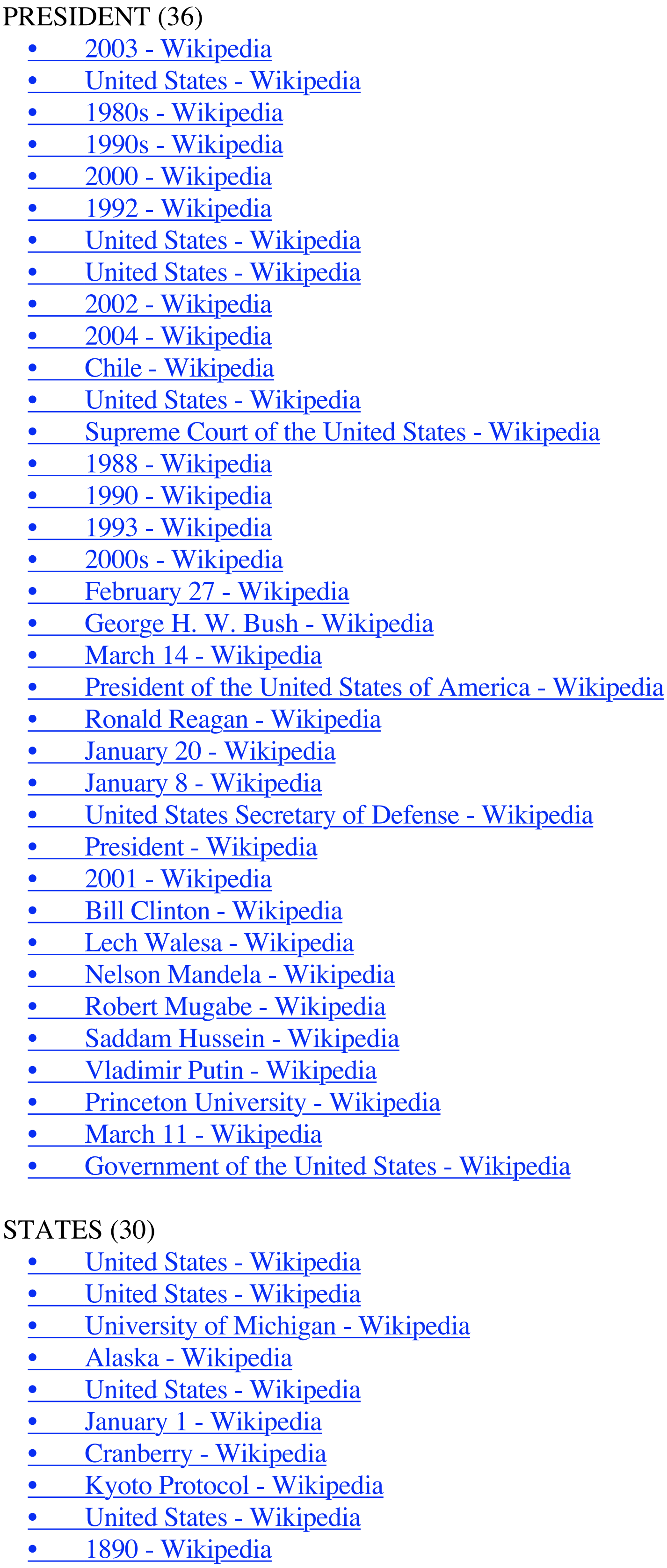


Appendix K

Search Result Clustering Tests

K.1 Search Word: Bush

K.1.1 Clustering



PRESIDENT (36)

- 2003 - Wikipedia
- United States - Wikipedia
- 1980s - Wikipedia
- 1990s - Wikipedia
- 2000 - Wikipedia
- 1992 - Wikipedia
- United States - Wikipedia
- United States - Wikipedia
- 2002 - Wikipedia
- 2004 - Wikipedia
- Chile - Wikipedia
- United States - Wikipedia
- Supreme Court of the United States - Wikipedia
- 1988 - Wikipedia
- 1990 - Wikipedia
- 1993 - Wikipedia
- 2000s - Wikipedia
- February 27 - Wikipedia
- George H. W. Bush - Wikipedia
- March 14 - Wikipedia
- President of the United States of America - Wikipedia
- Ronald Reagan - Wikipedia
- January 20 - Wikipedia
- January 8 - Wikipedia
- United States Secretary of Defense - Wikipedia
- President - Wikipedia
- 2001 - Wikipedia
- Bill Clinton - Wikipedia
- Lech Walesa - Wikipedia
- Nelson Mandela - Wikipedia
- Robert Mugabe - Wikipedia
- Saddam Hussein - Wikipedia
- Vladimir Putin - Wikipedia
- Princeton University - Wikipedia
- March 11 - Wikipedia
- Government of the United States - Wikipedia

STATES (30)

- United States - Wikipedia
- United States - Wikipedia
- University of Michigan - Wikipedia
- Alaska - Wikipedia
- United States - Wikipedia
- January 1 - Wikipedia
- Cranberry - Wikipedia
- Kyoto Protocol - Wikipedia
- United States - Wikipedia
- 1890 - Wikipedia

- Supreme Court of the United States - Wikipedia
- 1988 - Wikipedia
- Afghanistan - Wikipedia
- February 27 - Wikipedia
- Florida - Wikipedia
- George H. W. Bush - Wikipedia
- March 14 - Wikipedia
- President of the United States of America - Wikipedia
- January 20 - Wikipedia
- January 8 - Wikipedia
- March 28 - Wikipedia
- United States Secretary of Defense - Wikipedia
- President - Wikipedia
- Lech Walesa - Wikipedia
- Princeton University - Wikipedia
- March 11 - Wikipedia
- Aleutian Islands - Wikipedia
- Texas - Wikipedia
- Enron Corporation - Wikipedia
- Government of the United States - Wikipedia

HUMAN (9)

- Human cloning - Wikipedia
- Walking - Wikipedia
- Uneconomic growth - Wikipedia
- Mythical beast - Wikipedia
- Productivism - Wikipedia
- 21st century - Wikipedia
- Kennewick Man - Wikipedia
- 21st century - Wikipedia
- Nudity - Wikipedia

ACTOR (24)

- 1989 - Wikipedia
- 2003 - Wikipedia
- 2000 - Wikipedia
- 1992 - Wikipedia
- 2002 - Wikipedia
- January 1 - Wikipedia
- 1890 - Wikipedia
- 1988 - Wikipedia
- 1990 - Wikipedia
- 1993 - Wikipedia
- August 24 - Wikipedia
- February 27 - Wikipedia
- January 3 - Wikipedia
- March 14 - Wikipedia
- January 13 - Wikipedia
- January 20 - Wikipedia

- January 6 - Wikipedia
- January 8 - Wikipedia
- March 28 - Wikipedia
- United States Secretary of Defense - Wikipedia
- 2001 - Wikipedia
- Audrey Hepburn - Wikipedia
- July 6 - Wikipedia
- March 11 - Wikipedia

TREE (7)

- Ecology of Africa - Wikipedia
- Gooseberry - Wikipedia
- Shrub - Wikipedia
- Olive - Wikipedia
- Elephant - Wikipedia
- Eucalyptus - Wikipedia
- Koala - Wikipedia

TAX (9)

- Biosafety protocol - Wikipedia
- Earth Summit 2002 - Wikipedia
- Kyoto Protocol - Wikipedia
- Florida - Wikipedia
- Ronald Reagan - Wikipedia
- Margaret Thatcher - Wikipedia
- Dividend tax - Wikipedia
- Taxation in the United States - Wikipedia
- Insurance of terrorism - Wikipedia

UNIVERSITY (11)

- Manhattan Project - Wikipedia
- George Lakoff - Wikipedia
- University of Michigan - Wikipedia
- Alaska - Wikipedia
- Billy Graham - Wikipedia
- Florida - Wikipedia
- George H. W. Bush - Wikipedia
- Nelson Mandela - Wikipedia
- Princeton University - Wikipedia
- Massachusetts Institute of Technology - Wikipedia
- Texas - Wikipedia

NATIONAL (15)

- Alaska - Wikipedia
- Earth Summit 2002 - Wikipedia
- Kyoto Protocol - Wikipedia
- Chile - Wikipedia
- Billy Graham - Wikipedia
- President - Wikipedia

- Nickname - Wikipedia
- Lech Walesa - Wikipedia
- Nelson Mandela - Wikipedia
- Robert Mugabe - Wikipedia
- Saddam Hussein - Wikipedia
- Vladimir Putin - Wikipedia
- Audrey Hepburn - Wikipedia
- Government of the United States - Wikipedia
- Madrid Agreement - Wikipedia

PARTY (11)

- 1989 - Wikipedia
- Worldwide green parties - Wikipedia
- August 24 - Wikipedia
- George H. W. Bush - Wikipedia
- President of the United States of America - Wikipedia
- Ronald Reagan - Wikipedia
- President - Wikipedia
- Bill Clinton - Wikipedia
- Margaret Thatcher - Wikipedia
- Nelson Mandela - Wikipedia
- Wilhelm Reich - Wikipedia

Misc. (18)

- 1945 - Wikipedia
- Anti-globalization movement - Wikipedia
- Emu - Wikipedia
- Accounting scandals - Wikipedia
- Intellectual capital - Wikipedia
- Magic and religion - Wikipedia
- Magic and religion - Wikipedia
- Magic (paranormal) - Wikipedia
- Culture of Australia - Wikipedia
- Atheism - Wikipedia
- Alternative hip hop - Wikipedia
- The Simpsons - Wikipedia
- Smallpox - Wikipedia
- Australian Magpie - Wikipedia
- Behavioral finance - Wikipedia
- Consumerism - Wikipedia
- Monetarism - Wikipedia
- Magic (paranormal) - Wikipedia

K.1.2 Relevant vs Irrelevant Pages

RELEVANT

- Cranberry - Wikipedia
- Ecology of Africa - Wikipedia
- Gooseberry - Wikipedia
- Shrub - Wikipedia
- Olive - Wikipedia
- Elephant - Wikipedia
- Eucalyptus - Wikipedia
- Koala - Wikipedia
- Australian Magpie - Wikipedia
- Emu - Wikipedia

IRRELEVANT

- 2003 - Wikipedia
- United States - Wikipedia
- 1980s - Wikipedia
- 1990s - Wikipedia
- 2000 - Wikipedia
- 1992 - Wikipedia
- 2002 - Wikipedia
- 2004 - Wikipedia
- Chile - Wikipedia
- Walking - Wikipedia
- Supreme Court of the United States - Wikipedia
- 1988 - Wikipedia
- 1990 - Wikipedia
- 1993 - Wikipedia
- Mythical beast - Wikipedia
- 2000s - Wikipedia
- Aleutian Islands - Wikipedia
- February 27 - Wikipedia
- George H. W. Bush - Wikipedia
- March 14 - Wikipedia
- Alaska - Wikipedia
- President of the United States of America - Wikipedia
- Ronald Reagan - Wikipedia
- January 20 - Wikipedia
- January 8 - Wikipedia
- United States Secretary of Defense - Wikipedia
- President - Wikipedia
- 2001 - Wikipedia
- Bill Clinton - Wikipedia
- Lech Walesa - Wikipedia
- Nelson Mandela - Wikipedia
- Robert Mugabe - Wikipedia
- Saddam Hussein - Wikipedia
- Vladimir Putin - Wikipedia
- Princeton University - Wikipedia

- March 11 - Wikipedia
- Government of the United States - Wikipedia
- University of Michigan - Wikipedia
- January 1 - Wikipedia
- Kyoto Protocol - Wikipedia
- 1890 - Wikipedia
- Afghanistan - Wikipedia
- Florida - Wikipedia
- March 28 - Wikipedia
- Texas - Wikipedia
- Enron Corporation - Wikipedia
- Human cloning - Wikipedia
- Walking - Wikipedia
- Uneconomic growth - Wikipedia
- Productivism - Wikipedia
- 21st century - Wikipedia
- Kennewick Man - Wikipedia
- Nudity - Wikipedia
- 1989 - Wikipedia
- August 24 - Wikipedia
- January 3 - Wikipedia
- January 13 - Wikipedia
- January 6 - Wikipedia
- Audrey Hepburn - Wikipedia
- July 6 - Wikipedia
- Biosafety protocol - Wikipedia
- Earth Summit 2002 - Wikipedia
- Margaret Thatcher - Wikipedia
- Dividend tax - Wikipedia
- Taxation in the United States - Wikipedia
- Insurance of terrorism - Wikipedia
- Manhattan Project - Wikipedia
- George Lakoff - Wikipedia
- Billy Graham - Wikipedia
- Massachusetts Institute of Technology - Wikipedia
- Nickname - Wikipedia
- Lech Walesa - Wikipedia
- Madrid Agreement - Wikipedia
- Worldwide green parties - Wikipedia
- Wilhelm Reich - Wikipedia
- 1945 - Wikipedia
- Anti-globalization movement - Wikipedia
- Accounting scandals - Wikipedia
- Intellectual capital - Wikipedia
- Magic and religion - Wikipedia
- Magic (paranormal) - Wikipedia
- Culture of Australia - Wikipedia
- Atheism - Wikipedia
- Alternative hip hop - Wikipedia

- The Simpsons - Wikipedia
- Smallpox - Wikipedia
- Behavioral finance - Wikipedia
- Consumerism - Wikipedia
- Monetarism - Wikipedia

K.2 Search Word: Cd

K.2.1 Clustering

RECORD (17)

- Pink Floyd - Wikipedia
- Information science glossary of terms - Wikipedia
- The Scorpions - Wikipedia
- Cassette culture - Wikipedia
- Boy - Wikipedia
- Fletcher Henderson - Wikipedia
- 3 Feet High and Rising - Wikipedia
- The Beatles - Wikipedia
- Jimi Hendrix - Wikipedia
- The Beatles - Wikipedia
- Tom Waits - Wikipedia
- Yes - Wikipedia
- Compact disc - Wikipedia
- Spike Jones - Wikipedia
- Progressive rock - Wikipedia
- Album - Wikipedia
- CD-R - Wikipedia

TABLE (17)

- Electron affinity - Wikipedia
- Electronegativity - Wikipedia
- Periodic table - Wikipedia
- Electronegativity - Wikipedia
- Periodic table/Standard Table - Wikipedia
- ISO 3166-2 - Wikipedia
- Electronegativity - Wikipedia
- Periodic table/Electron configurations - Wikipedia
- Isotope table (complete) - Wikipedia
- Isotope table (divided) - Wikipedia
- Period 5 element - Wikipedia
- Extended periodic table - Wikipedia
- Periodic table/Alternate Table - Wikipedia
- Periodic table/Big Table - Wikipedia
- Periodic table/Huge Table - Wikipedia
- Periodic table/Metals and Non Metals - Wikipedia
- Periodic table/Wide Table - Wikipedia

SYSTEM (7)

- Functional decomposition - Wikipedia
- 21st century - Wikipedia
- Radar - Wikipedia
- 21st century - Wikipedia
- Federal Standard 1037C - Wikipedia
- Roman numerals - Wikipedia
- Roman numerals - Wikipedia

RIGHT (4)

- Copyright - Wikipedia

- Railroad switch - Wikipedia
- Wikipedia:History standards - Wikipedia
- Copy protection - Wikipedia

METAL (14)

- Electron affinity - Wikipedia
- Periodic table - Wikipedia
- Periodic table/Standard Table - Wikipedia
- Zinc - Wikipedia
- CD-R - Wikipedia
- Transition metal - Wikipedia
- Periodic table/Electron configurations - Wikipedia
- Cadmium - Wikipedia
- Period 5 element - Wikipedia
- Periodic table/Alternate Table - Wikipedia
- Periodic table/Big Table - Wikipedia
- Periodic table/Huge Table - Wikipedia
- Periodic table/Metals and Non Metals - Wikipedia
- Periodic table/Wide Table - Wikipedia

OPTICAL (5)

- Compact disc - Wikipedia
- Laser - Wikipedia
- Circular dichroism - Wikipedia
- Magnetic Circular Dichroism - Wikipedia
- CD-R - Wikipedia

BOOK (4)

- Oxford English Dictionary - Wikipedia
- Information science glossary of terms - Wikipedia
- The Lord of the Rings - Wikipedia
- British Library - Wikipedia

STANDARD (6)

- Wikipedia:History standards - Wikipedia
- Periodic table/Standard Table - Wikipedia
- ISO 3166-2 - Wikipedia
- Credit card - Wikipedia
- Federal Standard 1037C - Wikipedia
- ISO 3166-1 - Wikipedia

ATOMIC (5)

- Periodic table - Wikipedia
- SI base unit - Wikipedia
- Transition metal - Wikipedia
- Periodic table/Electron configurations - Wikipedia
- Cadmium - Wikipedia

Misc. (5)

- SI derived unit - Wikipedia
- Democratic Republic of the Congo - Wikipedia
- Romania - Wikipedia
- Candela - Wikipedia
- Star catalogue - Wikipedia

K.2.2 Relevant vs Irrelevant Pages

RELEVANT

- Electron affinity - Wikipedia
- Electronegativity - Wikipedia
- Periodic table - Wikipedia
- Periodic table/Standard Table - Wikipedia
- Periodic table/Electron configurations - Wikipedia
- Isotope table (complete) - Wikipedia
- Isotope table (divided) - Wikipedia
- Period 5 element - Wikipedia
- Periodic table/Alternate Table - Wikipedia
- Periodic table/Big Table - Wikipedia
- Periodic table/Huge Table - Wikipedia
- Periodic table/Metals and Non Metals - Wikipedia
- Periodic table/Wide Table - Wikipedia
- Zinc - Wikipedia
- Transition metal - Wikipedia
- Periodic table/Electron configurations - Wikipedia
- Cadmium - Wikipedia

IRRELEVANT

- Pink Floyd - Wikipedia
- Information science glossary of terms - Wikipedia
- The Scorpions - Wikipedia
- Cassette culture - Wikipedia
- Boy - Wikipedia
- Fletcher Henderson - Wikipedia
- 3 Feet High and Rising - Wikipedia
- The Beatles - Wikipedia
- Jimi Hendrix - Wikipedia
- Tom Waits - Wikipedia
- Yes - Wikipedia
- Compact disc - Wikipedia
- Spike Jones - Wikipedia
- Progressive rock - Wikipedia
- Album - Wikipedia
- CD-R - Wikipedia
- ISO 3166-2 - Wikipedia
- Functional decomposition - Wikipedia
- 21st century - Wikipedia
- Radar - Wikipedia
- Federal Standard 1037C - Wikipedia
- Roman numerals - Wikipedia
- Copyright - Wikipedia
- Railroad switch - Wikipedia
- Wikipedia:History standards - Wikipedia
- Copy protection - Wikipedia
- Laser - Wikipedia
- Circular dichroism - Wikipedia

- Oxford English Dictionary - Wikipedia
- The Lord of the Rings - Wikipedia
- British Library - Wikipedia
- Credit card - Wikipedia
- ISO 3166-1 - Wikipedia
- SI base unit - Wikipedia
- SI derived unit - Wikipedia
- Democratic Republic of the Congo - Wikipedia
- Romania - Wikipedia
- Candela - Wikipedia
- Star catalogue - Wikipedia

K.3 Search Word: Cluster

K.3.1 Clustering

GALAXIES (31)

- Galaxy formation and evolution - Wikipedia
- Timeline of knowledge about galaxies, clusters of galaxies, and large-scale structure - Wikipedia
- X-ray astronomy - Wikipedia
- Dark matter - Wikipedia
- Modified Newtonian dynamics - Wikipedia
- Computer cluster - Wikipedia
- New General Catalogue - Wikipedia
- Comoving distance - Wikipedia
- Dark matter - Wikipedia
- Large-scale structure of the cosmos - Wikipedia
- Lynx (constellation) - Wikipedia
- Virgo - Wikipedia
- Messier object - Wikipedia
- Globular cluster - Wikipedia
- Groups and clusters of galaxies - Wikipedia
- Supercluster - Wikipedia
- Triangulum Galaxy - Wikipedia
- Virgo Supercluster - Wikipedia
- Open cluster - Wikipedia
- 1 E23 m - Wikipedia
- Local Group - Wikipedia
- Virgo Supercluster - Wikipedia
- Active galaxy - Wikipedia
- Active galaxy - Wikipedia
- Drake equation - Wikipedia
- Active galaxy - Wikipedia
- Groups and clusters of galaxies - Wikipedia
- Large-scale structure of the cosmos - Wikipedia
- M87 - Wikipedia
- M87 - Wikipedia
- Virgo cluster - Wikipedia

STAR (25)

- X-ray astronomy - Wikipedia
- Modified Newtonian dynamics - Wikipedia
- Ursa Major - Wikipedia
- Stellar classification - Wikipedia
- Ara - Wikipedia
- Bayer designation - Wikipedia
- Ursa Major - Wikipedia
- Cancer (constellation) - Wikipedia
- Canis Major - Wikipedia
- Centaurus - Wikipedia
- Coma Berenices - Wikipedia
- Gemini - Wikipedia
- Monoceros - Wikipedia
- Scorpius - Wikipedia

- Serpens - Wikipedia
- Taurus - Wikipedia
- Virgo - Wikipedia
- Vulpecula - Wikipedia
- Protostar - Wikipedia
- Globular cluster - Wikipedia
- Open cluster - Wikipedia
- Drake equation - Wikipedia
- Blue straggler - Wikipedia
- Stellar classification - Wikipedia
- SETI - Wikipedia

CELLS (8)

- Pregnancy - Wikipedia
- Choanoflagellate - Wikipedia
- Genome - Wikipedia
- Morphogenesis - Wikipedia
- Streptococcus - Wikipedia
- MHC - Wikipedia
- MHC - Wikipedia
- Proteinoid - Wikipedia

TEST (6)

- Parapsychology - Wikipedia
- Race and intelligence - Wikipedia
- Stratified sampling - Wikipedia
- Survey sampling - Wikipedia
- Parapsychology - Wikipedia
- Severe acute respiratory syndrome - Wikipedia

VALUE (6)

- Argument - Wikipedia
- Gold - Wikipedia
- Drake equation - Wikipedia
- Seaweed - Wikipedia
- Market segment - Wikipedia
- Accuracy - Wikipedia

COMPUTER CPU MACHINE (5)

- Computer architecture - Wikipedia
- Distributed computing - Wikipedia
- Computer cluster - Wikipedia
- Non-Uniform Memory Access - Wikipedia
- Supercomputer - Wikipedia

MOLECULES (6)

- Lipid - Wikipedia
- Photosynthesis - Wikipedia
- Protostar - Wikipedia

- MHC - Wikipedia
- MHC - Wikipedia
- Hydrophobe - Wikipedia

MASS (13)

- Archipelago - Wikipedia
- Protostar - Wikipedia
- Groups and clusters of galaxies - Wikipedia
- Supercluster - Wikipedia
- Virgo Supercluster - Wikipedia
- Open cluster - Wikipedia
- Virgo Supercluster - Wikipedia
- Blue straggler - Wikipedia
- Groups and clusters of galaxies - Wikipedia
- M87 - Wikipedia
- M87 - Wikipedia
- Virgo cluster - Wikipedia
- Ion implantation - Wikipedia

FAMILY (5)

- Yarrow - Wikipedia
- Rambutan - Wikipedia
- MHC - Wikipedia
- MHC - Wikipedia
- Market segment - Wikipedia

Misc. (14)

- Weapon - Wikipedia
- Gender - Wikipedia
- Economies of agglomeration - Wikipedia
- Transistor - Wikipedia
- Weapon - Wikipedia
- Nord - Wikipedia
- Claude LÃ©vi-Strauss - Wikipedia
- Vietnam War - Wikipedia
- Milos - Wikipedia
- Cosmogony - Wikipedia
- Very Large Telescope - Wikipedia
- Very Large Telescope - Wikipedia
- Very Large Telescope - Wikipedia
- Crystallographic defects - Wikipedia

K.3.2 Relevant vs Irrelevant Pages

RELEVANT

- Computer cluster - Wikipedia
- Computer architecture - Wikipedia
- Distributed computing - Wikipedia
- Non-Uniform Memory Access - Wikipedia
- Supercomputer - Wikipedia

IRRELEVANT

- Galaxy formation and evolution - Wikipedia
- Timeline of knowledge about galaxies, clusters of galaxies, and large-scale structure - Wikipedia
- X-ray astronomy - Wikipedia
- Dark matter - Wikipedia
- Modified Newtonian dynamics - Wikipedia
- New General Catalogue - Wikipedia
- Comoving distance - Wikipedia
- Large-scale structure of the cosmos - Wikipedia
- Lynx (constellation) - Wikipedia
- Virgo - Wikipedia
- Messier object - Wikipedia
- Globular cluster - Wikipedia
- Groups and clusters of galaxies - Wikipedia
- Supercluster - Wikipedia
- Triangulum Galaxy - Wikipedia
- Virgo Supercluster - Wikipedia
- Open cluster - Wikipedia
- 1 E23 m - Wikipedia
- Local Group - Wikipedia
- Active galaxy - Wikipedia
- Drake equation - Wikipedia
- M87 - Wikipedia
- Virgo cluster - Wikipedia
- Ursa Major - Wikipedia
- Stellar classification - Wikipedia
- Ara - Wikipedia
- Bayer designation - Wikipedia
- Cancer (constellation) - Wikipedia
- Canis Major - Wikipedia
- Centaurus - Wikipedia
- Coma Berenices - Wikipedia
- Gemini - Wikipedia
- Monoceros - Wikipedia
- Scorpius - Wikipedia
- Serpens - Wikipedia
- Taurus - Wikipedia
- Vulpecula - Wikipedia
- Protostar - Wikipedia
- Blue straggler - Wikipedia

- SETI - Wikipedia
- Pregnancy - Wikipedia
- Choanoflagellate - Wikipedia
- Genome - Wikipedia
- Morphogenesis - Wikipedia
- Streptococcus - Wikipedia
- MHC - Wikipedia
- Proteinoid - Wikipedia
- Parapsychology - Wikipedia
- Race and intelligence - Wikipedia
- Stratified sampling - Wikipedia
- Survey sampling - Wikipedia
- Severe acute respiratory syndrome - Wikipedia
- Argument - Wikipedia
- Gold - Wikipedia
- Seaweed - Wikipedia
- Market segment - Wikipedia
- Accuracy - Wikipedia
- Lipid - Wikipedia
- Photosynthesis - Wikipedia
- Hydrophobe - Wikipedia
- Archipelago - Wikipedia
- Ion implantation - Wikipedia
- Yarrow - Wikipedia
- Rambutan - Wikipedia
- Weapon - Wikipedia
- Gender - Wikipedia
- Economies of agglomeration - Wikipedia
- Transistor - Wikipedia
- Nord - Wikipedia
- Claude Levi-Strauss - Wikipedia
- Vietnam War - Wikipedia
- Milos - Wikipedia
- Cosmogony - Wikipedia
- Very Large Telescope - Wikipedia
- Crystallographic defects - Wikipedia

K.4 Search Word: Dwarf

K.4.1 Clustering

WHITE_DWARF (32)

- Stellar evolution - Wikipedia
- Stellar evolution - Wikipedia
- Timeline of white dwarfs, neutron stars, and supernovae - Wikipedia
- X-ray astronomy - Wikipedia
- Goat - Wikipedia
- Dwarf - Wikipedia
- Pygmy mythology - Wikipedia
- Role-playing bestiary - Wikipedia
- Supernova - Wikipedia
- Red dwarf - Wikipedia
- Sirius - Wikipedia
- 1 E7 m - Wikipedia
- Centaurus - Wikipedia
- Magellanic Clouds - Wikipedia
- Local Group - Wikipedia
- Supernova remnant - Wikipedia
- Supernova 1987a - Wikipedia
- Supernova remnant - Wikipedia
- Black dwarf - Wikipedia
- Brown dwarf - Wikipedia
- Degenerate matter - Wikipedia
- Pauli exclusion principle - Wikipedia
- Planetary nebula - Wikipedia
- White dwarf - Wikipedia
- Proxima Centauri - Wikipedia
- Large Magellanic Cloud - Wikipedia
- Cataclysmic variable star - Wikipedia
- Roche limit - Wikipedia
- Nova - Wikipedia
- Mira - Wikipedia
- Procyon - Wikipedia
- Fermi gas - Wikipedia

STAR (31)

- Star - Wikipedia
- Stellar evolution - Wikipedia
- Stellar evolution - Wikipedia
- Timeline of white dwarfs, neutron stars, and supernovae - Wikipedia
- X-ray astronomy - Wikipedia
- Modified Newtonian dynamics - Wikipedia
- Dwarf - Wikipedia
- Pygmy mythology - Wikipedia
- Supernova - Wikipedia
- Red dwarf - Wikipedia
- Sirius - Wikipedia
- Stellar classification - Wikipedia
- Alpha Centauri - Wikipedia
- Centaurus - Wikipedia

- Supernova remnant - Wikipedia
- Extrasolar planet - Wikipedia
- Gas giant - Wikipedia
- Nemesis (star) - Wikipedia
- Supernova remnant - Wikipedia
- Black dwarf - Wikipedia
- Brown dwarf - Wikipedia
- Planetary nebula - Wikipedia
- White dwarf - Wikipedia
- Proxima Centauri - Wikipedia
- Stellar classification - Wikipedia
- Cataclysmic variable star - Wikipedia
- Roche limit - Wikipedia
- Nova - Wikipedia
- Mira - Wikipedia
- Procyon - Wikipedia
- Fermi gas - Wikipedia

SPECIES (9)

- Wheat - Wikipedia
- Zoology - Wikipedia
- Goat - Wikipedia
- Hippocampus (fish) - Wikipedia
- Whale - Wikipedia
- Bilberry - Wikipedia
- Crocodile - Wikipedia
- Zoology - Wikipedia
- Cuckoo - Wikipedia

HUMAN (8)

- Dwarf - Wikipedia
- Mythical beast - Wikipedia
- Pygmy mythology - Wikipedia
- Role-playing bestiary - Wikipedia
- Zulu mythology - Wikipedia
- Primate - Wikipedia
- Primate - Wikipedia
- English plural - Wikipedia

FRUIT (6)

- Rabbit - Wikipedia
- Apple - Wikipedia
- Bilberry - Wikipedia
- Mango - Wikipedia
- Olive - Wikipedia
- Specific replant disease - Wikipedia

AFRICA (5)

- History of Africa - Wikipedia

- History of Africa - Wikipedia
- Zulu mythology - Wikipedia
- Crocodile - Wikipedia
- History of Africa - Wikipedia

ORDER (8)

- Whale - Wikipedia
- Crocodile - Wikipedia
- Primate - Wikipedia
- Primate - Wikipedia
- 1 E17 s - Wikipedia
- 1 E7 m - Wikipedia
- 1 E20 m - Wikipedia
- Cuckoo - Wikipedia

FISH (5)

- Hippocampus (fish) - Wikipedia
- Crocodile - Wikipedia
- Sport - Wikipedia
- English plural - Wikipedia
- Sport - Wikipedia

STATES (4)

- California - Wikipedia
- Degenerate matter - Wikipedia
- Pauli exclusion principle - Wikipedia
- April 20 - Wikipedia

Misc. (4)

- Mythology - Wikipedia
- Galaxy formation and evolution - Wikipedia
- Elizabeth I of England - Wikipedia
- Pablo Picasso - Wikipedia

K.4.2 Relevant vs Irrelevant Pages

RELEVANT

- Stellar evolution - Wikipedia
- Timeline of white dwarfs, neutron stars, and supernovae - Wikipedia
- X-ray astronomy - Wikipedia
- Supernova - Wikipedia
- Red dwarf - Wikipedia
- Sirius - Wikipedia
- 1 E7 m - Wikipedia
- Centaurus - Wikipedia
- Magellanic Clouds - Wikipedia
- Local Group - Wikipedia
- Supernova remnant - Wikipedia
- Supernova 1987a - Wikipedia
- Black dwarf - Wikipedia
- Brown dwarf - Wikipedia
- Degenerate matter - Wikipedia
- Pauli exclusion principle - Wikipedia
- Planetary nebula - Wikipedia
- White dwarf - Wikipedia
- Proxima Centauri - Wikipedia
- Large Magellanic Cloud - Wikipedia
- Cataclysmic variable star - Wikipedia
- Roche limit - Wikipedia
- Nova - Wikipedia
- Mira - Wikipedia
- Procyon - Wikipedia
- Fermi gas - Wikipedia
- Star - Wikipedia
- Modified Newtonian dynamics - Wikipedia
- Stellar classification - Wikipedia
- Alpha Centauri - Wikipedia
- Extrasolar planet - Wikipedia
- Gas giant - Wikipedia
- Nemesis (star) - Wikipedia
- 1 E17 s - Wikipedia
- 1 E20 m - Wikipedia
- Galaxy formation and evolution - Wikipedia

IRRELEVANT

- Goat - Wikipedia
- Dwarf - Wikipedia
- Pygmy mythology - Wikipedia
- Role-playing bestiary - Wikipedia
- Wheat - Wikipedia
- Zoology - Wikipedia
- Hippocampus (fish) - Wikipedia
- Whale - Wikipedia
- Bilberry - Wikipedia

- Crocodile - Wikipedia
- Cuckoo - Wikipedia
- Mythical beast - Wikipedia
- Zulu mythology - Wikipedia
- Primate - Wikipedia
- English plural - Wikipedia
- Rabbit - Wikipedia
- Apple - Wikipedia
- Mango - Wikipedia
- Olive - Wikipedia
- Specific replant disease - Wikipedia
- History of Africa - Wikipedia
- Sport - Wikipedia
- California - Wikipedia
- April 20 - Wikipedia
- Mythology - Wikipedia
- Elizabeth I of England - Wikipedia
- Pablo Picasso - Wikipedia

Appendix L

E-Mail to Test Participants

Dear Friends

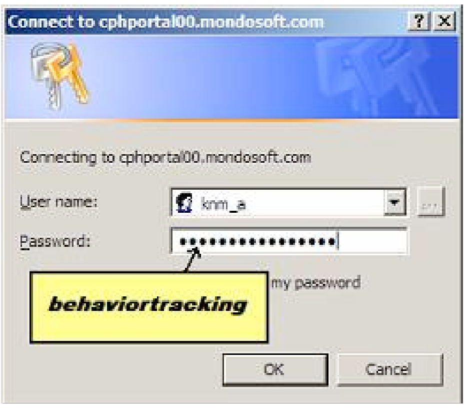
You are invited to participate in a user test of our “similar page categorisation” that we have been working on for Mondosoft for the past months as part of our master thesis. The duration of the test is up to you – you can end the test at any time, and even just a few inputs will help us a lot. You can also participate in the test several times if you are only able to spare 10 minutes now and then. Should you decide to participate in the test, please do so before July 15.

The test is about choosing which result group seems to be the most similar for a given page, which is presented to you. It is of course up to you to define for yourself what is similar and what is not, but the intention with the similar page system is to return links to pages that best seem to be in the same category as the original page. Using your input, we can hopefully prove that our system outperforms regular categorisation methods.

The test is located at http://clustering.mondosoft.com/

If you decide to participate, please follow the above link.

When asked to log in, enter *knm_a* as username and *behaviortracking* as password. This is also demonstrated in the below figure:



Connect to cphportal00.mondosoft.com

Connecting to cphportal00.mondosoft.com

User name: knm_a

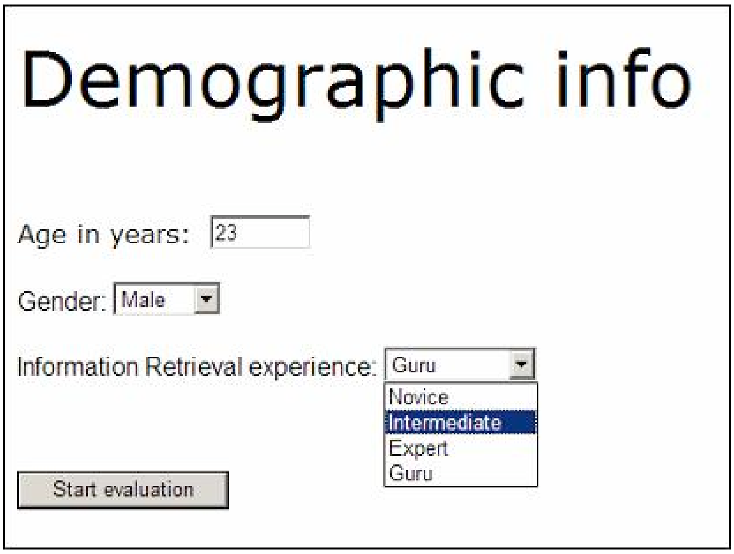
Password: ••••••••••••••••

behaviortracking

Hereafter, you are taken to a page, where you are asked to provide a little bit of information about yourself for demographic analysis.

The Information Retrieval experience falls into the following categories:
Novice: You have little or no experience using search engines (e.g. google)

Intermediate: You are using google or equivalent (e.g. yahoo) on a regular basis.
Expert: You have experience with multiple search engines, ability to perform advanced searches.
Guru: You have participated in creating a search engine.



Demographic info

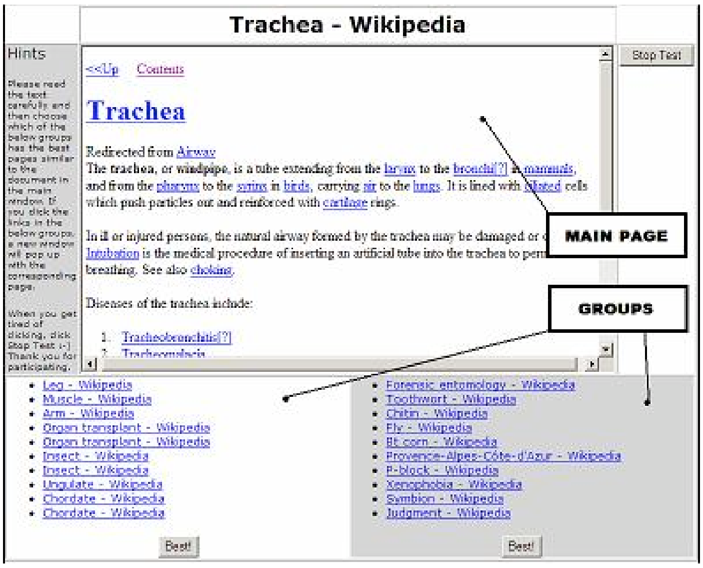
Age in years: 23

Gender: Male

Information Retrieval experience: Guru

Then, finally, you are presented with the actual test. As you can see in the below figure, the main page is in the middle of the window. Please take a moment to make sure that you understand what the article or page is about, and then look at the links in the groups below it.

When you click a link in any of the groups, the page corresponding to the link pops up and you can study it to see if you think it resembles the main page. (If you don’t see/get any pop-ups, you might need to enable pop-ups for this site.) When you have decided which group is the better one, please click the “Best!” button for the best group. If you are not sure which group to choose – just choose a random group.



Trachea - Wikipedia

Hints

Please read the text carefully and then choose which of the below groups has the best pages similar to the document in the main window. If you click the links in the below groups, a new window will pop up with the corresponding page.

When you get tired of clicking, click Stop Test :-) Thank you for participating.

<<Up Contents

Trachea

Redirected from Airway
The trachea, or windpipe, is a tube extending from the larynx to the bronchi[?] in mammals, and from the pharynx to the syrinx in birds, carrying air to the lungs. It is lined with ciliated cells which push particles out and reinforced with cartilage rings.

In ill or injured persons, the natural airway formed by the trachea may be damaged or c
Intubation is the medical procedure of inserting an artificial tube into the trachea to perm
breathing. See also choking.

Diseases of the trachea include:

1. Tracheobronchitis[?]
2. Tracheomalacia

- Leg - Wikipedia
- Muscle - Wikipedia
- Arm - Wikipedia
- Organ transplant - Wikipedia
- Organ transplant - Wikipedia
- Insect - Wikipedia
- Insect - Wikipedia
- Ungulate - Wikipedia
- Chordate - Wikipedia
- Chordate - Wikipedia

- Forensic entomology - Wikipedia
- Toothwort - Wikipedia
- Chitin - Wikipedia
- Fly - Wikipedia
- Bt corn - Wikipedia
- Provence-Alpes-Côte-d'Azur - Wikipedia
- P-block - Wikipedia
- Xenophobia - Wikipedia
- Symbion - Wikipedia
- Judgment - Wikipedia

To end the test, simply click the “Stop Test” button in the top corner.

If you experience any difficulties or bugs while testing, don’t hesitate to contact us (klv@mondosoft.com / martiny@mac.com) – we appreciate *any* form of feedback J

Thanks in advance,

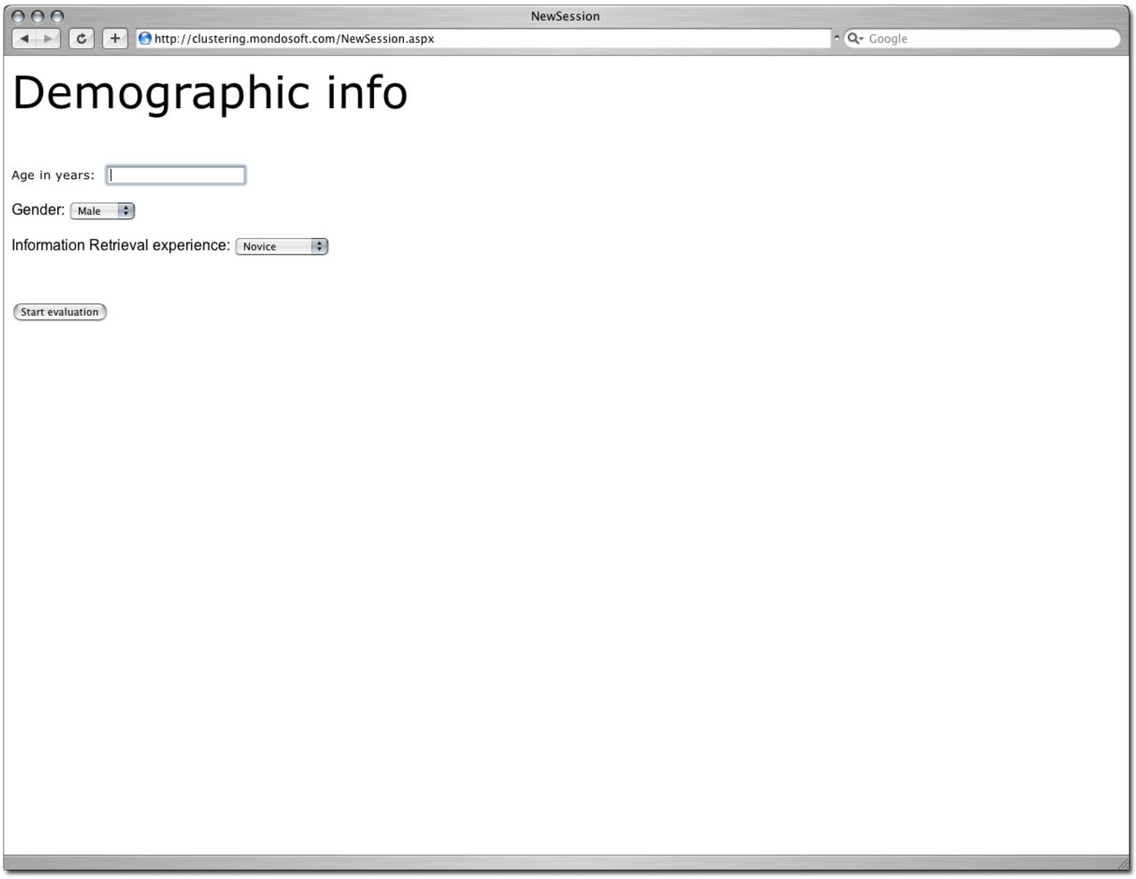
Moses Claus Martiny
M.Sc. Student
Email: martiny@mac.com
Phone: +45 3990 8494

Kenneth Lolk Vester
M.Sc. Student
Email: klv@mondosoft.com
Phone: +45 2834 2432

Appendix M

Online User Test Screenshots

M.1 Initial Demographic Questionnaire



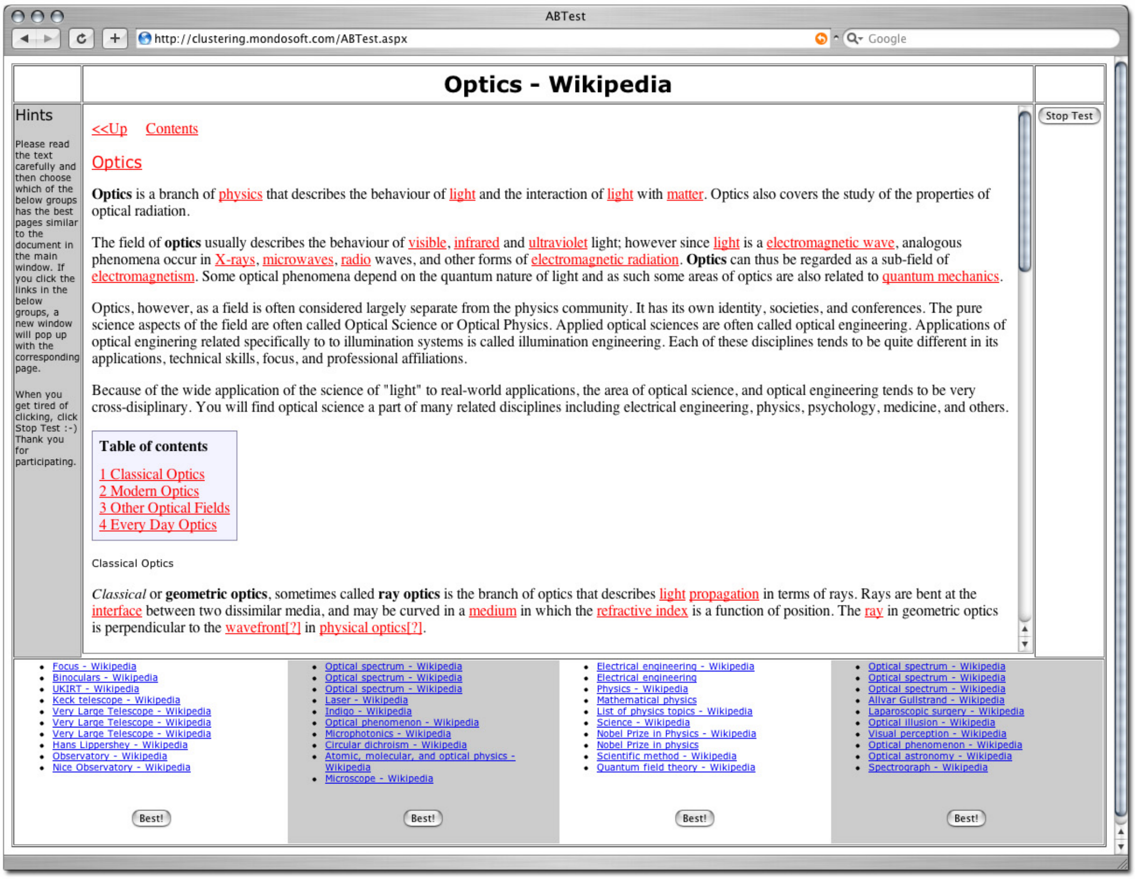
Demographic info

Age in years:

Gender: Male

Information Retrieval experience: Novice

M.2 Actual Test



Optics - Wikipedia

Hints

Please read the text carefully and then choose which of the below groups has the best pages similar to the document in the main window. If you click the links in the below groups, a new window will pop up with the corresponding page.

When you get tired of clicking, click Stop Test :-) Thank you for participating.

<<Up Contents

Optics

Optics is a branch of physics that describes the behaviour of light and the interaction of light with matter. Optics also covers the study of the properties of optical radiation.

The field of **optics** usually describes the behaviour of visible, infrared and ultraviolet light; however since light is a electromagnetic wave, analogous phenomena occur in X-rays, microwaves, radio waves, and other forms of electromagnetic radiation. **Optics** can thus be regarded as a sub-field of electromagnetism. Some optical phenomena depend on the quantum nature of light and as such some areas of optics are also related to quantum mechanics.

Optics, however, as a field is often considered largely separate from the physics community. It has its own identity, societies, and conferences. The pure science aspects of the field are often called Optical Science or Optical Physics. Applied optical sciences are often called optical engineering. Applications of optical enginering related specifically to to illumination systems is called illumination engineering. Each of these disciplines tends to be quite different in its applications, technical skills, focus, and professional affiliations.

Because of the wide application of the science of "light" to real-world applications, the area of optical science, and optical engineering tends to be very cross-disiplinary. You will find optical science a part of many related disciplines including electrical engineering, physics, psychology, medicine, and others.

Table of contents

Classical Optics

Classical or **geometric optics**, sometimes called **ray optics** is the branch of optics that describes light propagation in terms of rays. Rays are bent at the interface between two dissimilar media, and may be curved in a medium in which the refractive index is a function of position. The ray in geometric optics is perpendicular to the wavefront[?] in physical optics[?].

- Focus - Wikipedia
- Binoculars - Wikipedia
- UKIRT - Wikipedia
- Keck telescope - Wikipedia
- Very Large Telescope - Wikipedia
- Very Large Telescope - Wikipedia
- Very Large Telescope - Wikipedia
- Hans Lippershey - Wikipedia
- Observatory - Wikipedia
- Nice Observatory - Wikipedia

- Optical spectrum - Wikipedia
- Optical spectrum - Wikipedia
- Optical spectrum - Wikipedia
- Laser - Wikipedia
- Indigo - Wikipedia
- Optical phenomenon - Wikipedia
- Microphotonics - Wikipedia
- Circular dichroism - Wikipedia
- Atomic, molecular, and optical physics - Wikipedia
- Microscope - Wikipedia

- Electrical engineering - Wikipedia
- Electrical engineering
- Physics - Wikipedia
- Mathematical physics
- List of physics topics - Wikipedia
- Science - Wikipedia
- Nobel Prize in Physics - Wikipedia
- Nobel Prize in physics
- Scientific method - Wikipedia
- Quantum field theory - Wikipedia

- Optical spectrum - Wikipedia
- Optical spectrum - Wikipedia
- Optical spectrum - Wikipedia
- Allvar Gullstrand - Wikipedia
- Laparoscopic surgery - Wikipedia
- Optical illusion - Wikipedia
- Visual perception - Wikipedia
- Optical phenomenon - Wikipedia
- Optical astronomy - Wikipedia
- Spectrograph - Wikipedia