NTNU
Norwegian University of
Science and Technology

Dokument-klynging (document clustering)

Magnus Galåen

Master of Science in Informatics
Submission date: June 2008
Supervisor: Kjetil Nørvåg, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Problem Description

Klynging (clustering) er ein teknikk som kan brukast for å finne relaterte data. Det finnest mange algoritmer for klynging av generelle data, men ikkje alle av desse er eigna for klynging av dokument. Oppgåva går ut på å studere eigenskaper ved forskjellige algoritmer brukt til dokument-klynging i store dokumentsamlingar (inkl. web-data) og undersøkje tilhøyrande teknikkar som kan auke kvalitet på klynginga. Det er også ønskjeleg at aspekt rundt parallellisering/distribuering av klyngingsprosessen vert undersøkt.

Abstract

As document searching becomes more and more important with the rapid growth of document bases today, document clustering also becomes more important. Some of the most commonly used document clustering algorithms today, are pure statistical in nature. Other algorithms have emerged, adressing some of the issues with numerical algorithms, claiming to be better.

This thesis compares two well-known algorithms: Elliptic K-Means and Suffix Tree Clustering. They are compared in speed and quality, and it is shown that Elliptic K-Means performs better in speed, while Suffix Tree Clustering (STC) performs better in quality. It is further shown that STC performs better using small portions of relevant text (snippets) on real web-data compared to the full document. It is also shown that a threshold value for base cluster merging is unneccesary.

As STC is shown to perform adequately in speed when running on snippets only, it is concluded that STC is the better algorithm for the purpose of search results clustering.

Preface

This thesis is the final part of my masters degree on artificial intelligence in the Department of Computer and Information Science at the Norwegian University of Science and Technology.

My interest for categorization and clustering techniques began in 2002, during the course *machine learning*, in which we used Mitchell's book [Mitchell, 1997]. One specific memory is the day we read section 6.9: ”Naive Bayes Classifier”. I was really excited, and had many ideas. Like, how about automatic spam detection? At the time, spam was usually detected by handwritten rules, while today, spam learning techniques like the bayes classifier are considered a bare minimum of any spam filter.

One day, in another class, we were asked for new ideas regarding search engines. At once I raised my hand and suggested the use of automatic classification or clustering. Little did I know that this has been a part of information retrieval science since the earliest implementations. Today I am part of a start-up company specializing in information retrieval, where are lucky enough to be able to implement our ideas from the field.

I would like to thank my colleagues in SearchDaimon AS, my previous teachers at NTNU, and my supervisor. Thanks for all the fish!

Magnus Galåen

Contents

I Introduction 1

1 Introduction 3

II Theory 5

2 Information Retrieval 7
- 2.1 History 7
- 2.2 Inverted Index 7
- 2.3 The Vector Space Model 8

3 Document Preprocessing 11
- 3.1 Lexical Analysis 11
  - 3.1.1 Parsing of HTML 12
- 3.2 Stopwords 12
- 3.3 Stemming 13
- 3.4 Generating Snippets 13

4 Clustering 15
- 4.1 History 15
- 4.2 Numerical Clustering Algorithms 15
  - 4.2.1 Agglomerative Hierarchical Clustering 15
  - 4.2.2 K-Means 16
  - 4.2.3 Number of Clusters 17
- 4.3 Elliptic K-Means Clustering 17
- 4.4 Sparse Matrices 18

5 Suffix Tree Clustering 19
- 5.1 Introduction 19
- 5.2 Theoretical Background 19
  - 5.2.1 Suffix Trees 19

5.2.2 Generalized Suffix Trees 20
5.3 The Algorithm 22
5.3.1 Identifying Base Clusters 22
5.3.2 Combining Base Clusters 24
5.3.3 Presentation 24

III Implementation 25

6 The Information Retrieval System 27
6.1 Introduction 27
6.2 The Open Directory Project 28
6.3 Indexing 28
6.4 Search 29
6.5 Stemming 29
6.6 Parsing 30
6.6.1 process_doc 30
6.6.2 Snippets 30
6.6.3 Paragraph Filtering 32

7 Spherical K-Means 33
7.1 Gmeans 33
7.2 Running 33

8 STC 35
8.1 Input 35
8.2 Score 37
8.3 Threshold 37

9 Distributed Suffix Tree Clustering 39
9.1 Introduction 39
9.2 Implementation 39

IV Results 43

10 Cluster Quality 45
10.1 Quality Measure 45
10.2 Open Directory 46
10.3 Human-based Clustering 46
10.4 Results 47
10.4.1 Similarity Values 47

10.4.2 Removal or Keeping of Common Words 49

11 Speed Measure 53

11.0.3 stc_net 54

11.0.4 gmeans 54

12 Further Discussion and Conclusion 57

12.1 STC and Number of Base Clusters 57

12.2 STC and Hierarchical Clusters 58

12.3 Conclusion 60

A Programs 61

B Screenshots of Browser Representation 63

List of Figures

3.1 Snippet example 14

4.1 AHC dendrogram 16
4.2 K-Means visualization of clusters 17

5.1 Suffix tree for the string *bananas* 20
5.2 Suffix tree with suffix pointers 21
5.3 Generalized suffix tree 23

10.1 Similarity values, a comparision 50

11.1 Total running time based on document set size. With snippets. 54
11.2 Total running time based on document set size. With paragraphs. 55

12.1 Similarity values with different thresholds. 59

B.1 Snapshot 1 64
B.2 Snapshot 2 65

List of Tables

3.1 Stemming example 13

6.1 Inverted index storage structure 29
6.2 Morphology pseudo-code 31

8.1 STC document record 36

9.1 Distributed STC pseudo-code 41

10.1 Manually constructed cluster labels 47
10.2 STC (with snippets) 48
10.3 STC (with paragraphs) 49
10.4 Generated cluster labels with common words removed. 51
10.5 Generated cluster labels without common words removed. . . 52

11.1 Number of documents and words 53

12.1 Top clusters for two different threshold values. 58
12.2 Threshold and number of clusters. 58

Part I

Introduction

Chapter 1

Introduction

Every day new documents are created. The number of digital documents in the world has a exponential growth. Search engines does a great job by making these documents easily available to the world population. Documents get ranked by popularity, which are great, when you know how to define a good query. But, alas, most people today specify only one term when querying for information. When searching for general topics, it gets harder to find what you are searching for.

Automatic Clustering done at search time, aims at identifying groups, or clusters, of documents, and present the user with a list of possible subqueries to narrow down the search. In this way, any user can find what they want in a much easier way. For instance a query for ”car” might give clusters like ”buy car”, ”car rental” or ”comparisons of cars”. The user can then easily pin down the query.

At least that is how it is supposed to work. Document Clustering today may produce some good clusters, but often present irrelevant ones, too. Clustering search engines don’t even report which algorithms they use. This thesis aims at comparing two of the popular clustering algorithms in terms of quality and speed. Different parameters and input data are tested, to see if this can improve the quality, and a distributed version of one of them is implemented, in order to test for speed gains.

To make reading easier, whenever this thesis refers to ”we”, it actually only refers to the author.

Part II

Theory

Chapter 2

Information Retrieval

2.1 History

Automatic information retrieval as a concept was first introduced in 1945 in the ground breaking article ”As We May Think” (see [Bush, 1996] for a reprint). Later, the idea of indexing documents into words was described by [Luhn, 1957]. And in the 1960s, the SMART system was developed [Salton, 1971]. But until the 1990s, information retrieval systems were only used on small document collections. Therefore major development was made when the Text Retrieval Conference introduced large document collections for use in research [Harman, 1993], and later with the introduction of web search engines [Lewis, 1995].

For a more detailed overview of the history of Information Retrival, the reader is encouraged to read [Singhal, 2001].

2.2 Inverted Index

An inverted index is a sorted list of words, with the list of corresponding documents attached to each word. Given documents d_0 = ”Jupiter and Mars are gods from ancient greece”, d_1 = ”Venus and Jupiter are the brightest planets in the night sky” and d_2 = ”The sky is dark in the night”, one might construct an inverted index as follows:

token	list of occurences
ancient	$d_0[6]$
and	$d_0[1], d_1[1]$
are	$d_0[3], d_1[3]$
brightest	$d_1[5]$
dark	$d_2[3]$
from	$d_0[5]$
gods	$d_0[4]$
greece	$d_0[7]$
in	$d_1[7], d_2[4]$
is	$d_2[2]$
jupiter	$d_0[0], d_1[2]$
mars	$d_0[2]$
night	$d_1[9], d_2[6]$
planets	$d_1[6]$
sky	$d_1[10], d_2[1]$
the	$d_1[4], d_1[8], d_2[0], d_2[5]$
venus	$d_1[0]$

If the user wants all documents containing the word ”jupiter”, it’s only a matter of looking up ”jupiter” in the table and collect the documents (d_0 and d_1). Or, if the user wants all documents containing both words ”dark” and ”night”, the system will look up both words and merge them together. Only documents occuring in both lists will be kept (the *union* of documents). In the latter example, only document d_2 will be retrieved. Document positions are often kept for phrase or proximity searches. For example, only d_1 contains the words ”night sky” in sequence (called a phrase).

Words occuring in many documents, such as ”the”, are usually removed and marked as stopwords. Stopwords take up a lot of space, and don’t give much meaning to a document.

2.3 The Vector Space Model

The Vector Space Model was introduced in [Salton et al., 1975]. Documents are represented as vectors in a multidimensional Euclidean space, where each axis corresponds to a term [Chakrabarti, 2003]. The model has been criticized for being *ad hoc* [Raghavan and Wong, 1986]. It has been, and is still widely used in information retrieval.

For the purpose of information retrieval and document clustering, a term or token refers to a word. The weight for term t_i in document d_j is defined as:

$$w_{i,j} = tf \cdot idf \tag{2.1}$$

Where tf is the *term frequency*, and idf is the *inverse document frequency*. Different formulas may be applied to calculate these values. In our implementation, we ended up using the formulas from the SMART system (where the Vector Space Model was first implemented), as these seemed to give the best results at the time. $n_{i,j}$ is the number of times term t_i occurs in document d_j, and D_i is the set of documents containing t_i.

$$tf_{i,j} = \begin{cases} 0 & \text{if } n_{i,j} = 0 \\ 1 + \log(1 + \log(n_{i,j})) & \text{otherwise} \end{cases} \tag{2.2}$$

$$idf_i = \log \frac{1 + |D|}{|D_i|} \tag{2.3}$$

Given a query q, we construct a vector $\vec{q}$. The degree of similarity between document d_j and query q may be calculated, for instance, by the cosine of the angle between the two vectors [Baeza-Yates et al., 1999]:

$$sim(d_j, q) = \frac{\vec{d_j} \cdot \vec{q}}{|\vec{d_j}| \times |\vec{q}|} = \frac{\sum_{i=1}^{t} w_{i,j} \times w_{i,q}}{\sqrt{\sum_{i=1}^{t} w_{i,j}^2} \times \sqrt{\sum_{j=1}^{t} w_{i,q}^2}} \tag{2.4}$$

The same similarity measure may be used to calculate the similarity between two documents.

Chapter 3

Document Preprocessing

This chapter presents several key issues in document preprocessing. [Baeza-Yates et al., 1999] have divided document preprocessing into several text operations. Here are some of them:

1. Lexical analysis with the objective to transform raw data into seqences of letters (words).

2. Marking of stopwords with the objective to filter out very common words from subsequent processing.

3. Stemming/lemmatization of the remaining words with the objective to recognize similar words in different tenses.

The result can be a list of keywords for indexing. Or it can be a snippet for the user to view. Context defines the use.

3.1 Lexical Analysis

Lexical analysis is the process of transforming a sequence of characters into a sequence of words. Given a document d, different uses of lexical analysis may be:

1. Transforming document d into the sequence of all words contained in the document.

2. Transforming document d into the sequence of all text paragraphs contained in the document.

3. Given a query q, transforming d into a snippet containing as many as possible of the words in q.

3.1.1 Parsing of HTML

Most documents available for the public today, can be found on the world wide web. And most of these follow the HTML or XHTML standards [Raggett et al., 1999].

A naïve parser implementation is to remove all tags and keep all words. A more advanced parser has several additional considerations:

1. How to handle different codesets. Languages such as japanese and russian use different encodings. Even the latin alfabet may be represented using several codesets. A solution is to convert all characters into utf-8.
2. Html escapes, such as &aring;, may also represent characters. These should also be converted to utf-8.
3. Html comments should be removed.
4. Some text might be invisible, and should be removed. This is mostly needed in the case of spam-websites.
5. Partition text into headers, paragraphs, subparagraphs, and mark end-of-sentences.

Web documents contains much noise in forms of hypertext links, tables, menus, and listing of non-english tokens. [Chudnovsky, 2005] suggests several methods to filter out this noise. One way to do this, is only keeping text paragraphs that contain meaningful words and few links. Linktext (anchored text) are usually more relevant to the document they point to, rather than the document in which they are contained[1].

3.2 Stopwords

There are several ways to create a list of stopwords. One way is to get hold of a list of common words defined in the english dictionary. Another strategy is to build one by taking the n most common words from the inverted index. The latter will contain words specific for the document base. For internet documents this might include words as: *copyright, information, home* and *web*.

It is not always an advantage to remove stopwords completely. Although they are seldom included in the inverted index, documents with stopwords

[1] in fact, several search engines make an index over linktext, so that even if the keyword isn't explicitly contained in the document, it may be in the linktext of pages linking to it

Table 3.1: Stemming: Some words and their resulting base forms using WordNet's morphological function.

angels	→ angel	crawling	→ crawl	went	→ go
running	→ run	brainier	→ brainy	goes	→ go
ran	→ run	better	→ good	chairs	→ chair
indices	→ index	explore	→ explore		

removed make strange snippets. Stopwords are rather kept in a separate list, and marked when encountered. Algorithms running calculations will then ignore them unless they are needed.

3.3 Stemming

Stemming and lemmatization are two related processes of word simplification. Lemmatization is the process of reducing the different senses of a word to a common lemma. Stemming is the process of reducing a word to its stem. But while lemmatization requires an advanced dictionary to ensure correctness, the stem doesn't need to be identical with the morphological root. In fact, the stem can be practically anything as long as all inflections of the word are reduced to the same stem.

A stemmer can be implemented with a simple algorithm, such as the Porter Stemming Algorithm [Porter, 1980]. But it is not always correct. Both ”university” and ”universal” are transformed to ”univers” when using Porter. An hybrid approach is to use dictionaries for exceptions from the rules (ran → run). WordNet uses this approach [Fellbaum et al., 1998].

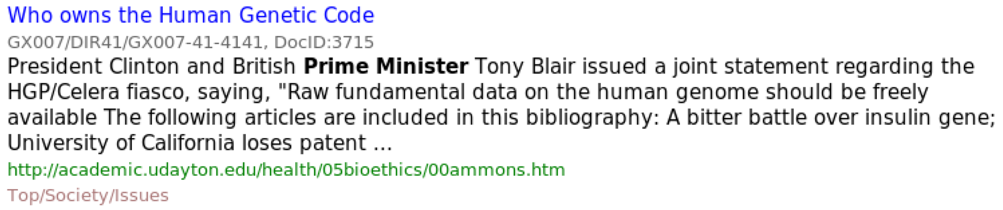
3.4 Generating Snippets

A snippet is defined as a window or a glimpse into the text, which will give the user an idea about the document contents. The snippet is not a part of document preprocessing itself, but combined with a query, a product of it. Figure 3.1 gives an example of a snippet generated by our IR (Information Retrieval) system.

Given a document d and a query q, finding the best snippet $s(d, q)$ may be seen as the process of calculating all possible snippets, giving each of them a score, and picking the best one. The score can be calculated based on the following criteria:

1. Should contain as many query keywords as possible.

Figure 3.1: Snippet example



Who owns the Human Genetic Code

GX007/DIR41/GX007-41-4141, DocID:3715

President Clinton and British **Prime Minister** Tony Blair issued a joint statement regarding the HGP/Celera fiasco, saying, "Raw fundamental data on the human genome should be freely available The following articles are included in this bibliography: A bitter battle over insulin gene; University of California loses patent ...

http://academic.udayton.edu/health/05bioethics/00ammons.htm

Top/Society/Issues

2. Should give meaningful sentences.

3. Preferably at beginning of a paragraph or a meaning.

4. If several snippets give the same score, pick the first one.

If it is difficult to fullfill the first criteria, is is possible to generate two queries of half the size, and concatenate them with ”...”. Snippet generation may be implemented efficiently using dynamic programming.

Chapter 4

Clustering

4.1 History

Clustering algorithms in Information Retrieval can be traced back to [Jardine and van Rijsbergen, 1971] and even [Broffit et al., 1966]. The Cluster Hypothesis states that similar documents tend to be relevant to the same queries. [Van Rijsbergen, 1979] suggest that clusters be computed once for the entire corpora, so that documents from the precomputed clusters get included in the retrieval phase. However, [Hearst and Pedersen, 1996] revises this view, and states that if two documents d_1 and d_2 are both relevant or not relevant for a given query, they must not be both relevent or nonrelevant for a different query. Today, clustering techniques are usually applied at search time.

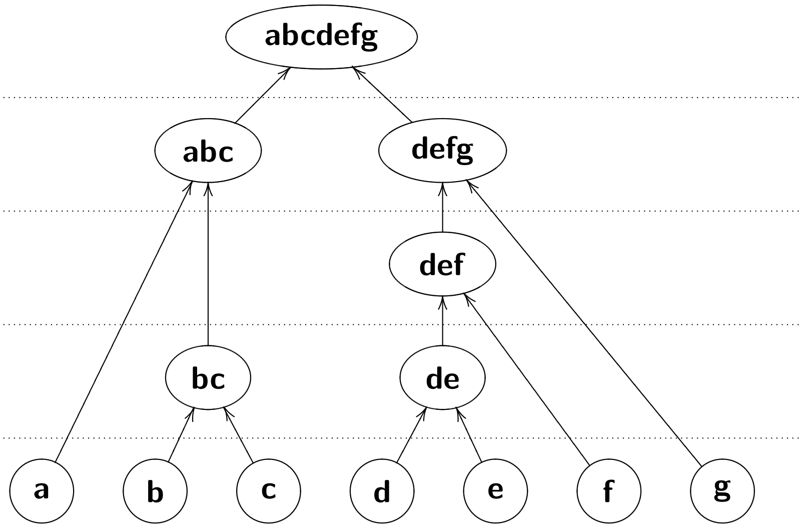
4.2 Numerical Clustering Algorithms

Agglomerative Hierarchical Clustering (AHC) has been widely used in document clustering and other IR applications for a long time [Rasmussen, 1992, Willett, 1988]. Today AHC and K-Means clustering are regarded as the two main approaches to document clustering in literature [Steinbach et al., 2000]. The Scatter/Gather document browsing system implements a combination of both to "get the best of both worlds" [Cutting et al., 1992].

4.2.1 Agglomerative Hierarchical Clustering

Also called *bottom-up* clustering, AHC begins with n clusters, where n is the number of documents. Clusters are then compared, and the clusters closest to eachother are merged. This is done in several steps, until all clusters are

Figure 4.1: This type of graph is called a dendrogram, and displays the merging of clusters a-g from bottom to top.



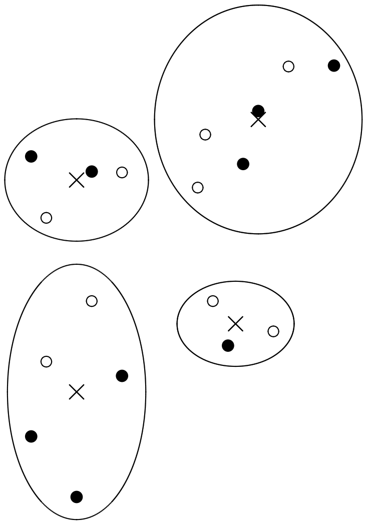
merged into one. See Figure 4.1. To get the desired number of clusters, the user will cut the tree at a given height. The comparison between clusters are performed using a similarity function $s(d_1, d_2)$, usually the $tf \cdot idf$ cosine measure [Chakrabarti, 2003].

4.2.2 K-Means

AHC is fairly easy to implement, but uses a total running time of $O(n^2 \log n)$. It is not practical for large document sets. A common alternative is the K-Means algorithm, first presented in [MacQueen, 1967]. The algorithm depends on the user to choose the number k different clusters in advance. As opposed to AHC, K-Means is a flat non-hierarchical algorithm, and runs in $O(kn)$ time.

Documents are represented as vectors. k initial *centroids*, one for each cluster, are either picked randomly, or generated from a heuristic. All document vectors are then compared to the centroids, and documents are assigned to the cluster of the *nearest* (most similar) centroid. The centroids are then recalculated to be the best possible representation of it's current cluster, and clusters are regenerated according to the new centroid values. This continues iteratively until the clusters or centroids doesn't change (much) [Chakrabarti, 2003].

Figure 4.2: A 2-dimensional representation of a clustergraph. Centroids are marked with an ×.



4.2.3 Number of Clusters

Both AHC and K-Means depends on the user to choose the number of desired clusters. A common choice is $k = \sqrt{N}$, where N is the number of documents. Another approach is *the elbow criterion*, which says that you should choose a number of clusters so that adding another cluster doesn't add sufficient information.

4.3 Elliptic K-Means Clustering

In the original K-Means algorithm, similarity between the centroid and a document is calculated simply using euclidean distance. However, for high-dimensional data such as text documents, represented as $tf \cdot idf$ vectors, cosine similarity has been shown to be a superior measure [Strehl et al., 2000]. This suggest that the direction of a document vector is more important than the magnitude [Zhong, 2005].

According to [Dhillon et al., 2001], elliptic K-Means has the property of being able to exploit the sparsity of the vector space model, and can use sparse matrices, as described in the next section. Elliptic K-Means runs in time and space linear to the number of documents.

4.4 Sparse Matrices

An ordinary matrix can be refered to as a *dense matrix*. All values are stored, column by column. Huge matrices takes up a lot of memory. Sparse matrices on the other hand, only store the non-zero values and the corresponding column and row indices. Dense matrices use $m \times n$ space, while sparse matrices use $3 \times nz$, where nz is the number of non-zero values.

This works well for document vectors. The total number of unique terms is at least an 6-digit order of magnitude, while a typical web-document seldom have more than a couple of thousand words. This means a lot of zeros that can be skipped.

In a matrix $m \times n$, with nz non-zero values, we define three arrays:

```
int col[n+1];
int row[nz];
double val[nz];
```

Column i starts at $col[i]$, and ends at $col[i+1]-1$. $col[n]$ equals nz.

$row[col[i]]$ is the row index of the 1st non-zero element in the i-th column.

$val[col[i]]$ is the value of the 1st non-zero element in the i-th column.

For example, consider the following matrix:

$$\begin{bmatrix} 0 & 0.25 & 0 & 0 & 0 & 0 & 0 \\ 0.25 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3.0 & 3.43 & 3.5 & 0 & 0 \\ 0 & 4.78 & 4.0 & 0 & 0 & 2.72 & 0 \\ 1.33 & 0 & 0 & 0 & 0 & 7.8 & 6.05 \end{bmatrix}$$

It may be stored as a sparse matrix in three arrays as follows:

col	0		2		4		6	7	8		10	11
row	1	4	0	3	2	3	2	2	3	4	4	
val	0.25	1.33	0.25	4.78	3.0	4.0	3.43	3.5	2.72	7.8	6.05	

Chapter 5

Suffix Tree Clustering

5.1 Introduction

In the traditional approach to document clustering, documents are treated as sets of words, disregarding word sequences. [Zamir et al., 1997] argues word proximity may be valuable in some cases, and furthermore that phrases makes cluster labels more human readable. Based on these findings, Suffix Tree Clustering (STC) was presented in [Zamir and Etzioni, 1998]. The assumption is that common topics often share identical phrases, and that phrases usually are more informative than unorganized sets of keywords. If two documents contain the phrase ”Ben and Jerry’s ice cream”, it can be assumed they are somehow relevant to eachother [Stefanowski and Weiss, 2003].

5.2 Theoretical Background

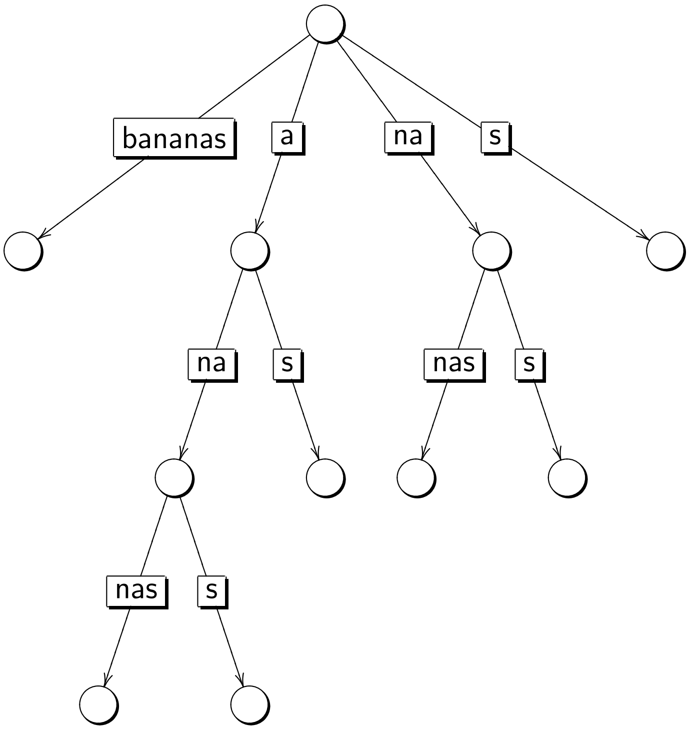
In this section, we present the basics of suffix trees and generalized suffix trees.

5.2.1 Suffix Trees

The suffix tree was first introduced by [Weiner, 1973]. It is a data structure which contains all the suffixes of a given string, so as to run many important string operations more efficiently. The string may be a string of characters (”c”,”a”,”t”), which we use in the following examples. In the STC implementation, however, we will use string of words (”cat”,”ate”,”fish”). The suffix tree for the string S is defined as a tree such that [Gusfield, 1997]:

- the paths from the root to the leaves have a one-to-one relationship with the suffixes of S.

Figure 5.1: The suffix tree for the string *bananas*.



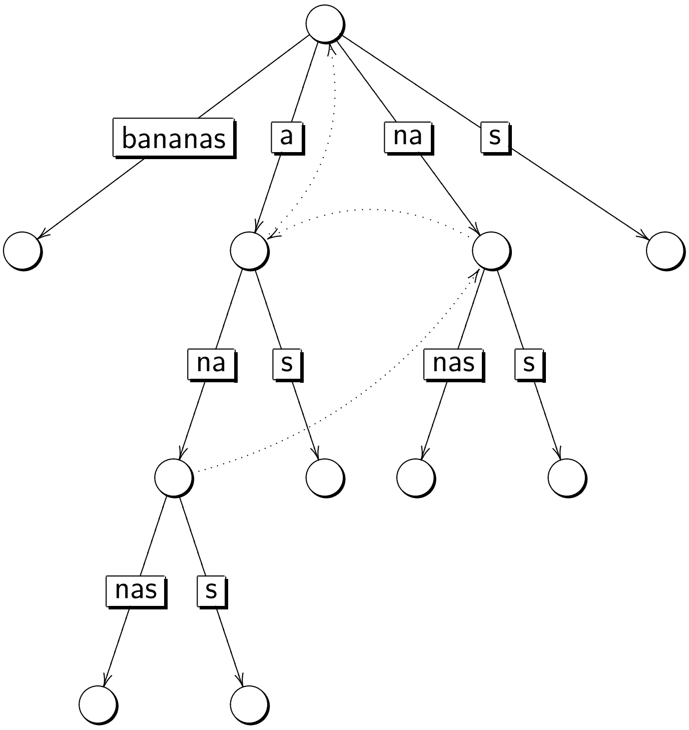
- all edges are labeled with non-empty strings.
- all internal nodes (except perhaps the root) have at least two children.

In this project we implemented suffix trees with Ukkonen's algorithm [Ukkonen, 1995]. Ukkonen's algorithm has linear time complexity in the length of the string. It also has the property of being on-line, which means it processes the string from left to right. Since leaf nodes never will become internal nodes, we only have to update the internal nodes for each element of the string. Each node represent a string $S[i..j]$. For each internal node, we assign a suffix pointer pointing to the *last* node of the suffix string $S[i+1..j]$. When adding a new element to the suffix tree, we only have to traverse the internal nodes through the suffix pointers, and thus the algorithm is linear in time [Nelson, 1996].

5.2.2 Generalized Suffix Trees

Ordinary suffix trees has the disadvantage of being able to only hold one string of elements. To be able to hold several strings, we have to construct a

Figure 5.2: The suffix tree for the string *bananas* with suffix pointers added.



data structure called a generalized suffix tree. This is possible in linear time [Bille, 2005]:

- Append all strings to one big string

$$C = S_1\$_1...S_n\$_n \tag{5.1}$$

- Build a suffix tree from C using Ukkonen's algorithm.
- Label all leaves by their positions in C.
- Remove all suffixes that span multiple strings.
- Convert leaf labels to include all strings containing the suffix.

5.3 The Algorithm

The STC algorithm consists of two major phases: the identifying and combining of base clusters.

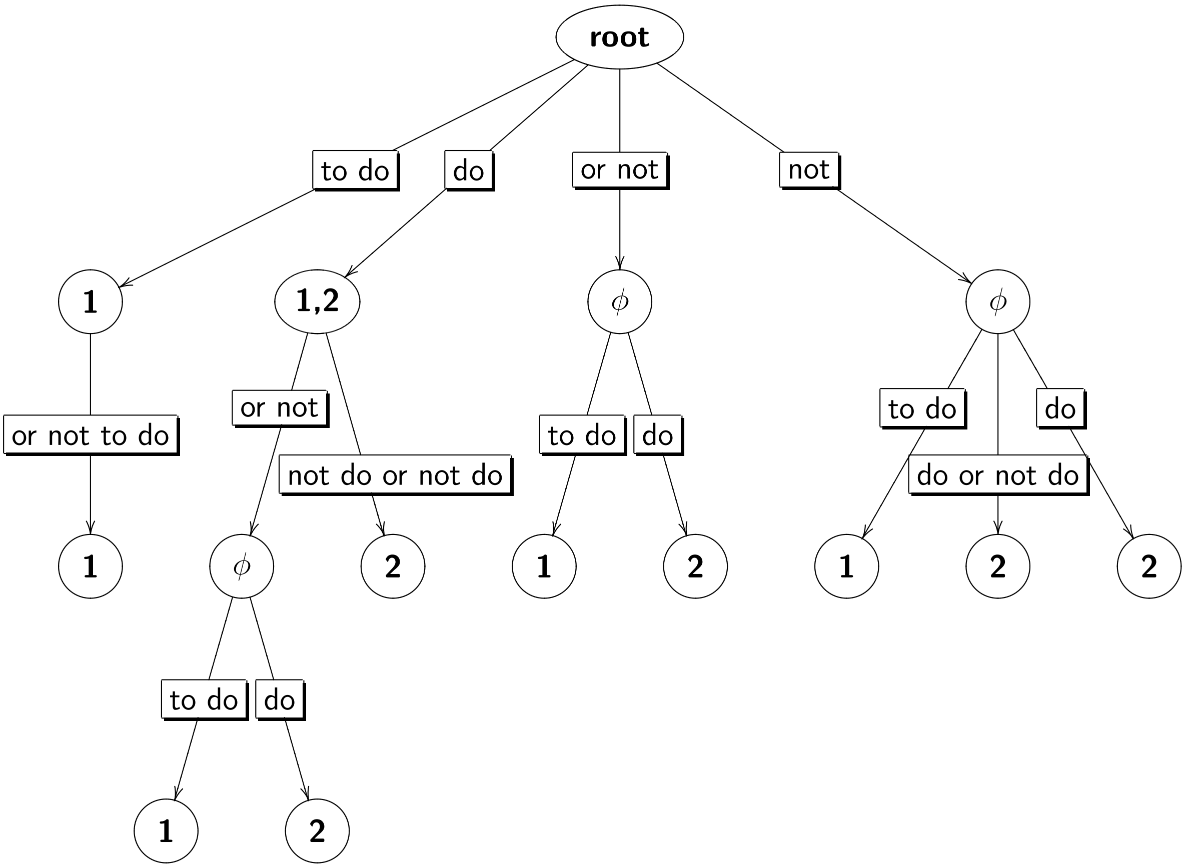
5.3.1 Identifying Base Clusters

After the documents have been cleaned and transformed into strings of words, they are all inserted into a generalized suffix tree. All nodes (both internal and external) in the tree with two or more documents, represent phrases common to all documents in the node. Therefore all such nodes are base clusters. Each base cluster is assigned a score

$$s(B) = |B| \cdot f(|P|) \tag{5.2}$$

where $|B|$ is the number of documents in the base cluster, and $|P|$ is the number of words in phrase P that have a non-zero score. [Zamir and Etzioni, 1998] states simply that words appearing in the stoplist, or that appear in too few (3 or less) or too many (more than 40%) of the documents, receive a score of zero, and all other words are not. <somebody> suggest each word can have a score according to it's df (document frequency) value. The function f penalizes phrases consisting of one word, gives an linear increasing value to phrases from two to six words, and a constant value for longer phrases.

Figure 5.3: The generalized suffix tree for the strings *"to do or not to do"* and *"do not do or not do"*



5.3.2 Combining Base Clusters

In this step, identical or similar base clusters are combined to form final clusters. Base clusters B_m and B_n are similar if and only if equation 5.3 and equation 5.4 holds. $|B_m \cap B_n|$ represents the number of documents common to both base clusters.

$$\frac{|B_m \cap B_n|}{|B_m|} > 0.5 \tag{5.3}$$

$$\frac{|B_m \cap B_n|}{|B_n|} > 0.5 \tag{5.4}$$

This forms a graph of connected components. Coherent subgraphs are merged into final clusters. To preserve the time linear property of the algorithm, [Zamir and Etzioni, 1998] perform this phase with only the k best base clusters ($k = 500$ in their experiments). Several other authors however, define a score threshold instead. [Stefanowski and Weiss, 2003] observes that the choice of threshold strongly affects the number of discovered clusters.

5.3.3 Presentation

[Zamir and Etzioni, 1998] show only the top 10 base clusters, and labels it with all of the phrases. They don't say, however, how they rate the clusters. It has been suggested using the combined score of all base clusters for this rating.

STC produces a flat structure of possibly overlapping clusters. Documents may share more than one phrase with each other, and may therefore belong to more than one cluster. This acknowledges the fact that documents may have more than one topic.

The STC algorithm has a theoretically linear time complexity on the number of words: $O(|W|)$.

Part III

Implementation

Chapter 6

The Information Retrieval System

6.1 Introduction

This chapter describes the implementation of the Information Retrieval System.

When reading reports on comparing or implementing clustering algorithms for the use in document clustering, usually two different forms of document collections are used. One is to use carefully crafted document sets, like the TREC Collections. These sets are built from newswire reports, abstracts of scientific articles from certain fields, journal articles, or even internet blogs. Each set has a strict structure which all documents follow, and are usually stored i XML or SGML format. One drawback of using this method, is that even if algorithms may perform well on these document sets, they may have major problems when facing real unstructured data.

The other form of document sets widely used, is simply to use or build a meta-search engine. Meta-search engines sends out queries to a number of internet search engines, and collocates the result snippets to form the data sets. But when grouping the top ten or twenty results from a number of internet search engines, the snippets may have an unusually high quality when compared to ordinary web-documents.

Part of this assignment was to run the clustering algorithms on web-data. We want to see how the clustering algorithms perform on ordinary, unstructured web-content. For this we built our own information retrieval system.

For convenience, the system has been split in several programs. This is because when doing research, we sometimes want to run the same subprocess over and over again, on the same data. The most important programs are:

`build`	Processes all files, and builds the inverted index.
`makeidf`	Calculates and stores the *idf* value for each word for efficient lookup.
`stoplist`	Extracts the n most common words from the inverted index.
`search`	Performs a search and returns a list of document id's.
`process_doc`	Retrieves a list of documents, and does the necessary preprocessing.
`showclusters`	Visualizes the documents in their corresponding clusters for easy browsing.

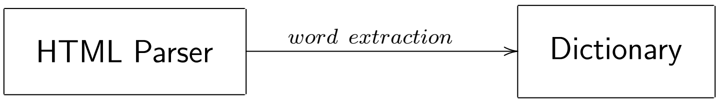
See Appendix A for a closer look at each program. Pictures from the resulting graphical view can be found in Appendix B.

6.2 The Open Directory Project

Web Crawlers are subject to a number of issues. They might follow looping links (documents refering to eachother), overwhelm servers with rapid requests, and download huge amounts of unwanted data. The internet is literally filled with spam. Web directories contain only manually added URLs, and they are even categorized according to topic. We downloaded approximately 600.000 documents referred to from the Open Directory Project (www.dmoz.org). By this, we avoided the potential problems of using a crawler, and did not have to worry about spam pages.

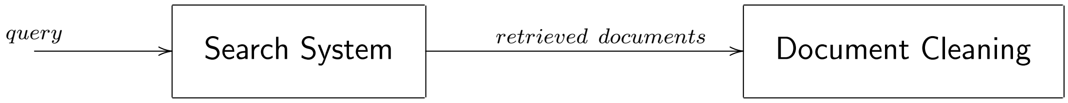
6.3 Indexing

After downloading the documents, we built an inverted index. All documents were parsed accoring to the rules in section 3.1.1. Words were extracted, and we built a dictionary containing all unique words. Each word in the dictionary contained a link to an static address in another file, listing the document id's for documents containing the word. See Table 6.1 for details.



At first we performed stemming, document cleaning, etc. at this stage (building time), but this soon turned out to be unflexible. The complete database had to be rebuilt even after minimal changes in the parser or stopword handling. These parts were moved to search time instead.

6.4 Search



The clustering algorithms were tested with several kinds of input data. This was handled by the search system. When retrieving documents from a given query, the users has to specify the whether to:

- Extract the full document, paragraphs only, or snippet only.
- Return results according to the vector space model (for G-Means) or as a list of words (for STC).
- Filter out numbers.
- Apply stemming.
- Use stopwords, and which sets of stopwords to use.

6.5 Stemming

Stemming was implemented with basis in WordNet's [Fellbaum et al., 1998] morphology function. Since WordNet is not optimized for this kind of use, it performed badly in terms of speed. So we did some changes:

1. All words defined in the WordNet corpora were extracted to build a simple and clean dictionary, containing only the words.
2. All words defined in the exception lists for the morphing functions were merged into one file, containing only one lemma and one tense per line.
3. The complete dictionary and exception list is read into memory at execution time.

Table 6.1: Structure of the database files used in the inverted index

`dictionary`	
string	word
pointer	ii_ptr

`ii`	
int	size
list<pointer>	id_ptr

`id`	
string	title
string	url
string	path
string	category

WordNet treats nouns, verbs, adjectives and adverbs separately. As our system does not know which class a word belongs to, we assume it's the first one that fits. The stemmer function *morph*(), and the related *defined*() function are shown in table 6.2.

6.6 Parsing

Lexical analysis was done using an HTML parser written in *lex* and *yacc*. After a search, the documents are retrieved using `process_doc`.

6.6.1 process_doc

`process_doc` takes as input a list of document id's. This program does a lot of the magic before the clustering process takes place. `process_doc` has the following parameter flags:

flags	
-means <prefix>	Notifies `process_doc` to create a sparse matrix in the CCS format as output (for `gmeans`). When not specified, the program will generate lists of words instead (for STC).
-snippet (default)	Only tokens in the snippet will be used.
-paragraph	Only tokens appearing in text paragraphs will be used.

The '-snippet' and '-paragraph'-flags may not be used at the same time.

6.6.2 Snippets

Snippets are generated in the following way:

1. The document is scanned for all words contained in the query.

2. Paragraph and sentence boundaries are marked positive.

3. Anchor text is marked negative.

For each possible snippet, points are given according the above criterias. The best snippet is returned. In some cases the snippet is split into two smaller snippets separated with '...'. Internet Search Engines usually calculate snippets of approximately 160 characters. For the purpose of clustering, more relevant information can be gained by increasing this size. Therefore snippet size has been set to be 320 characters wide.

Table 6.2: Pseudo C++-code for the morph() and defined() function.

```
bool defined( string s )
{
        if (Dictionary.find(s))
                return true;
        else
                return false;
}

string morph( string s )
{
        if (s.length()<=2) return s;

        if (it=ExceptionList.find(s))
                return (*it).second;

        string           sufx[20] = {
            /* Noun suffixes */
            "s", "ses", "xes", "zes", "ches", "shes", "men", "ies",
            /* Verb suffixes */
            "s", "ies", "es", "es", "ed", "ed", "ing", "ing",
            /* Adjective suffixes */
            "er", "est", "er", "est"
        };
        string           addr[20] = {
            /* Noun endings */
            "", "s", "x", "z", "ch", "sh", "man", "y",
            /* Verb endings */
            "", "y", "e", "", "e", "", "e", "",
            /* Adjective endings */
            "", "", "e", "e"
        };

        string  stem, suffix;

        if (is_suffix(s, "ful"))
            {
                stem.assign(s, 0, s.length()-3);
                suffix = "ful";
            }
        else
            {
                stem = s;
                suffix = "";
            }

        for (int i=0; i<20; i++)
            {
                if (is_suffix(stem, sufx[i]))
                    {
                        string  ret;
                        ret.assign(stem, 0, stem.length() - sufx[i].length());
                        ret+= addr[i];

                        if (defined(ret))
                            {
                                ret+= suffix;
                                return ret;
                            }
                    }
            }

        return s;
}
```

6.6.3 Paragraph Filtering

Paragraphs, or blocks of text within the document, are recognized by separating the stream of words with paragraph boundaries. To do this, we use the information already contained in the HTML. Tags like <p>, <div>, <table> and <tr>, marks local boundaries of text. For each paragraph, tokens are matched against the dictionary. If a paragraph contains 40 words or more, and at least 60% of the words exists in the dictionary, it is regarded it a informative paragraph and kept. All other text is filtered out.

Chapter 7

Spherical K-Means

7.1 Gmeans

For running the Spherical K-Means algorithm, the gmeans package was used. This package is freely available from *http://www.cs.utexas.edu/users/dml/Software/gmeans.html*. Gmeans is available under the GNU General Public License, and was written by Yuqiang Guan. Also see [Dhillon and Modha, 2001]. Some minor adjusting was done to the source code in order to make it work with the gcc 4.1 compiler.

Gmeans uses the *sparse matrix* as described in section 4.4. Following the CCS format, the matrix itself is spread among three files: matrix_col_ccs, matrix_row_ccs, and matrix_tfn_nz. In addition, matrix_docs lists the full document id's, and matrix_words lists the words.

After a succesful run, the results are written to files matrix_tfn_doctoclus.X and matrix_tfn_wordtoclus.X. The first file lists clusters and document id's. The latter lists the cluster each word belongs to. The big X is replaced by a number indicating the final number of clusters.

7.2 Running

When running gmeans, the user has to specify the number of clusters gmeans should generate. This can be a range. As default, gmeans will try to pick the 1st initializing centroids as far from the center of the data set as possible, and well separate all centroids. This is not always a good choice when dealing with document sets, so we used randomly picked vectors as centroids instead.

When running gmeans, we used the following command:

```
gmeans -i c -c <#clusters> -W -w -1 <prefix>
```

Where *prefix* is the prefix of the input files, and $\sqrt{|N|}$ was inserted as the clustersize. `gmeans` will not necessarily return the specified number of clusters, it is merely a suggestion.

Chapter 8

STC

Suffix Tree Clustering was implemented using the algorithms described in chapter 5.

8.1 Input

The input file (generated by `process_doc`) contains all the necessary document information needed. For each document, there is a record of the following format:

docid Docid of the retrieved document.

url Internet URL.

path Local path and filename.

category Category as listed in dmoz.

title Document title.

snippet Generated snippet.

data Sequence of words and *idf* values.

Table 8.1 shows a sample document. If, when running `process_doc`, the *paragraph* flag was specified, the 'data'-field will include all the paragraphs. The only restriction is that the field will be truncated if its size exceeds 1MB (which is *a lot* for a stripped html-file).

Table 8.1: A sample document retrieved from the query ”1929”. Each word in the ’data’-field is followed by its *idf* value. Stopwords recieve a score of zero.

docid 22577

url http://econ161.berkeley.edu/TCEH/Slouch_Crash14.html

path GX000/DIR98/GX000-98-23998

category Top/Society/History

title Sliding into the Great Depression

snippet The stock market did crash in October of **1929**; ”Black Tuesday”;, October 29, **1929**, saw American common stocks lose something like a tenth of their value. That it was ripe for a bursting of the bubble is well known; the exact reasons why the bubble burst then are unknowable; more important are the consequences of ...

data the 0 stock 1.14465 market 2.63389 do 0 crash 0.629546 october 0.238003 quot 0 black 0.0881097 tuesday 0.0997683 quot 0 october 0.238003 see 0 american 0.191818 common 0.110997 stock 1.14465 lose 0.675898 something 0 like 0 tenth 0 their 0 value 0 that 0 be 0 ripe 0 for 0 burst 0.61328 the 0 bubble 0.472029 well 0 know 0 the 0 exact 0 reason 0.245312 why 0 the 0 bubble 0.472029 burst 0.61328 then 0 be 0 unknowable 0 more 0 important 0.281325 be 0 the 0 consequence 0

8.2 Score

Two base cluster score schemes were tested. The first follows the scheme of the original article [Zamir and Etzioni, 1998]:

- Phrases of size 1 were given a score of 0.30.
- Phrases of sizes 2 to 6 were given a score of $1.75 + \frac{size}{2}$.
- Longer phrases were given a score of 4.75.

Additionaly, each stopword was rewarded with a score of 0.001, as phrases like ”the planet mars” are more descriptive than ”planet mars”. The score was finally multiplied with the number of documents the base cluster occured in.

[Stefanowski and Weiss, 2003] uses a variation, and includes the $tf \cdot idf$ score for each word. We slightly modified this, and made the following scheme:

- Let $x = \sum_w idf(w)$ for all words w in the phrase that is not a stopword.
- Phrases of size 1 were given a score of $0.30 + \frac{x}{10}$.
- Phrases of sizes 2 to 6 were given a score of $1.75 + \frac{size}{2} + \frac{x}{size \cdot 10}$.
- Longer phrases were given a score of $4.75 + \frac{x \cdot 6}{size \cdot 10}$.

As before, stopwords were rewarded with 0.001 score, and the final score was multiplied with the number of documents. The tf score was not included, as the term frequency of a random document containing the phrase of the basecluster should be irrelevant.

8.3 Threshold

Only the top base clusters are chosen for cluster calculation. The program supports both picking the top 500 (or any other value) base clusters, as in the original article, or the use of a threshold. When using a score threshold, every base cluster scoring above the threshold will be picked.

Chapter 9

Distributed Suffix Tree Clustering

9.1 Introduction

For this experiment we chose the STC algorithm. The on-line property of the algorithm and the way base clusters are found and combined, makes this a good candidate for an distributed implementation.

The documents are equally divided among the nodes. Each node performs the identifying phase, and score calculations independent of eachother. The master node does the final clustering, which is fast. All the time-consuming parts are distributed among the nodes.

9.2 Implementation

The STC source code was expanded to include server and client networking. When executing the server (or the master node), a prompt like this will appear:

```
Waiting for connections. Enter 'start' to begin.
New connection(0)
New connection(1)
New connection(2)
```

The client program can be run from any computer. No data files are necessary. Everything is transferred through the master node.

The user initializes the clustering process by typing start into the terminal. The master node will then read all document data according to Section

8.1. The data is split in equal sizes for all nodes. As the STC algorithm has a running time based on number of words, and not number of documents (as elliptic K-Means), this split will be performed based on document lengths.

The data is then sent to all client nodes, and each node (including the master) immediately starts processing. Each node goes through the indentifying phase on its own. Then they will all calculate base cluster score independent of each other (the first part of the second phase). The best base clusters will be sent back to the master node, which merges all the baseclusters. When the best base clusters of the merged list are chosen, requests are sent to all nodes to return the list of documents for all phrases contained in these. The final merging to clusters will be performed at the master node. The full data flow between the nodes can be seen in Table 9.1.

It is the users choice if the top 500 baseclusters should be picked, or if a threshold should be followed. In the case of a threshold value, each node will use the threshold value divided by the number of nodes. Because if a final base cluster scores above the threshold, at least one node should have the same base cluster scoring above the reduced threshold.

Table 9.1: C++ pseudo-code for the server-side client/server communication.

```
int n = server->numConnections();

for each client[c]
  client[c] << threshold / n;

for each document[d]
  for each {client[c], master node}
    if this node has recieved fewest words so far
      {
        if this node is a client
          client[c] << document[d];
        else
          SuffixTree.add_document(document[d]);

        break;
      }

for each client[c]
  client[c] << signal_no_more_documents;

base_clusters bc = SuffixTree.calculate_base_clusters();

for each client[c]
  {
    base_clusters c_bc;
    client[c] >> c_bc;
    merge c_bc into bc;
  }

base_clusters top = pick_best_baseclusters(b);

for each client[c]
  client[c] << top;

final_base_clusters fbc = SuffixTree.get_base_clusters(top);

for each client[c]
  {
    final_base_clusters c_fbc;
    client[c] >> c_fbc;
    merge c_fbc into fbc;
  }

results r = calculate_clusters(fbc);
print_results(r);
```

Part IV

Results

Chapter 10

Cluster Quality

How can we measure the quality of clusters? Evaluation of results is perhaps the most difficult part when testing search results clustering algorithms, according to [Stefanowski and Weiss, 2003]. [Macskassy et al., 1998] studied human based clustering, and found little similarity between different subjects. Nevertheless, a common approach is to use predefined collections and measure the quality with *information-theoretic entropy*.

10.1 Quality Measure

[Dom, 2001] has proposed a measure of quality between two partitions of a set of objects. In clustering algorithms we refer to the *ground truth* as the wanted partition. Several formulas are given for different cases. We need one which doesn't require the algorithmically generated partition to have the same number of clusters as the ground truth partition.

We applied the same measures as in [Stefanowski and Weiss, 2003]. From Byron Dom [Dom, 2001]:

$$Q_0 = -\sum_{c=1}^{|C|}\sum_{k=1}^{|K|}\frac{h(c,k)}{n}\log\frac{h(c,k)}{h(k)} + \frac{1}{n}\sum_{k=1}^{|K|}\log\binom{h(k)+|C|-1}{|C|-1} \tag{10.1}$$

C represents the *ground truth* partition, and K represents the partition generated by the cluster algorithm. c and k are the clusters of C and K respectively. $h(c,k)$ is the number of documents labeled cluster c that are assigned to k. $h(k)$ is $h(c,k)$ for all $c \in C$. n is the number of documents.

10.2 Open Directory

Since we only used documents from the Open Directory Project, all documents are already precategorized. However, when given a specific search query, we found that topics tended to be spread over several categories. For instance, the cluster ”artificial intelligence” belongs to Computers/Artificial_Intelligence, Games/Video_Games and Society/Philosophy. And some categories are found in nearly all clusters, like Regional/North_America.

First, instead of formulating a query, we extracted 3500 random documents from 7 categories (500 from each). These were then clustered using both `gmeans` and STC. Immediately this was shown to not be a good approach. Why? The most common phrases were phrases such as ”this page has moved” and ”copyright 2007”. Even with a big list of stopwords (3500 words) extracted from the inverted index, most clusters were based on similarities in structure, not in content. An even bigger list of stopwords may have been used, but then useful words started to disappear too.

This problem might be avoided by using document sets like the TREC collection, which has pure content. It is also possible to give negative score to all paragraphs which contain an sufficient amount of stopwords, and remove complete paragraphs. In either way, this is not really search results clustering (it is performed before search time), and is out of the scope for this thesis.

When using search and queries, content-free documents will (usually) not appear at all. And the documents will be more similar in general, giving more weight to topic-specific words, which in turn make the clusters. In addition, when using queries and snippets only, snippets only give words in proximity to the search query. Such words will seldom be of an unrelated topic. That is why the problems discussed in the previous paragraph seldom affect normal search results clustering.

10.3 Human-based Clustering

What we did was to conduct a series of queries, and manually cluster them based on 10-15 keyword phrases (subqueries). Several queries were used. Example queries shown in this thesis are:

- Harry Potter (673 documents).
- Prime Minister (1531 documents).
- Intelligence (4733 documents).

Table 10.1: Manually constructed cluster labels

Harry Potter	**Prime Minister**
lord of the rings	tony blair
philosopher's stone	gordon brown
chamber of secrets	election
prisoner of azkaban	parliament
goblet of fire	kevin rudd
order of the phoenix	stephen harp
half-blood prince	ehud olmert
deathly hallows	british
(harry potter) movie	deputy prime minister
j.k. rowling	ariel sharon
daniel radcliffe	
hogwarts	

Examples of the handpicked clusters are presented in table 10.1.

10.4 Results

The STC algorithm, with snippets as input, produced the clusters presented in table 10.2 when given queries "Harry Potter" and "Prime Minister". The clusters are represented by the phrase with the highest base cluster score. As the reader might notice, the words have been subject to stemming, and small words like "of" are not included at all.

As we can see, the cluster names give an good indication on the clusters produced. Several of the names suggest they represent the same clusters as the manually labeled ones.

However, STC with all text paragraphs as input, seems to produce clusters based on more general terms, as can be seen in the table below. While snippets only contains words in proximity of the query, the complete document might discuss the broader topic in general. Such as *politics* or the *european union* in the case of the query "Prime Minister" (in table 10.3).

10.4.1 Similarity Values

All queries were run through both the `gmeans` and the STC algorithm. Figure 10.1 show the similarity values between the produced clusters, and the ground truth. As we can see, STC produces more similar clusters given snippets only, while `gmeans` running elliptic K-Means produces more similar values when

Table 10.2: STC (with snippets)

Harry Potter
lord the ring
deathly hallow
chamber secret
half blood prince
prison azkaban
warn bros
mario matchmaking mariah
pirate the caribbean
the sorcerer's stone
philosopher's stone
clich philosophy plato's
the train layout
john harry grainger
phoenix
the goblet fire
relate topic biografi
rowling bloomsbury
harry potter character

Prime Minister
tony blair
constitution executive judicial
gordon brown
universit bern institut
kevin rudd
elect direct popular
stephen harp
ehud olmert
british
deputy
judicial court
universit bern institut
constitution
ariel sharon
appoint
manmohan singh
unicameral seat member
elect the president
israeli
cambodia nov cambodian
nawaz sharif
jawaharlal nehru
follow joint meet
pakistan continue concern
judicial court supreme
datebook november kyrgyz
relate taha ramadan's
call fax the
ben gurion
tony blair's
prime minister kevin
saxe coburg gotha
and imperil spokeswoman
freedom party announce
canadian prime minister
nov that form
look for right
israeli prime minister
there be pressure

Table 10.3: STC (with paragraphs)

Prime Minister
political
comoros congo brazzaville
soviet union
supreme court
bush administration
democratic
fund
europe
european union

given the bigger portion of information (paragraphs).

Several articles, among them [Zamir and Etzioni, 1998], have reported that using snippets for the clustering process, is ”good enough”. In our tests, we found that STC actually performs better using snippets, when used on real web-data.

10.4.2 Removal or Keeping of Common Words

We now remove words that occur more than 40% in the query-produced document set (in addition to stopwords). But what if we didn't? Given the query ”Intelligence”, STC produces the clusters in tables 10.4 and 10.5. As we can see, when the common words are kept, the description labels are more informative, and several of the top clusters (the lists are sorted) appears to be more relevant. But if we do the same test with ”Harry Potter”, it has a partial opposite effect. For instance, the top cluster is ”harry potter and”. The second cluster is ”the harry potter”, and the list includes seven more clusters of the form ”harry potter and <something>”.

Figure 10.1: Similarity values, a comparision

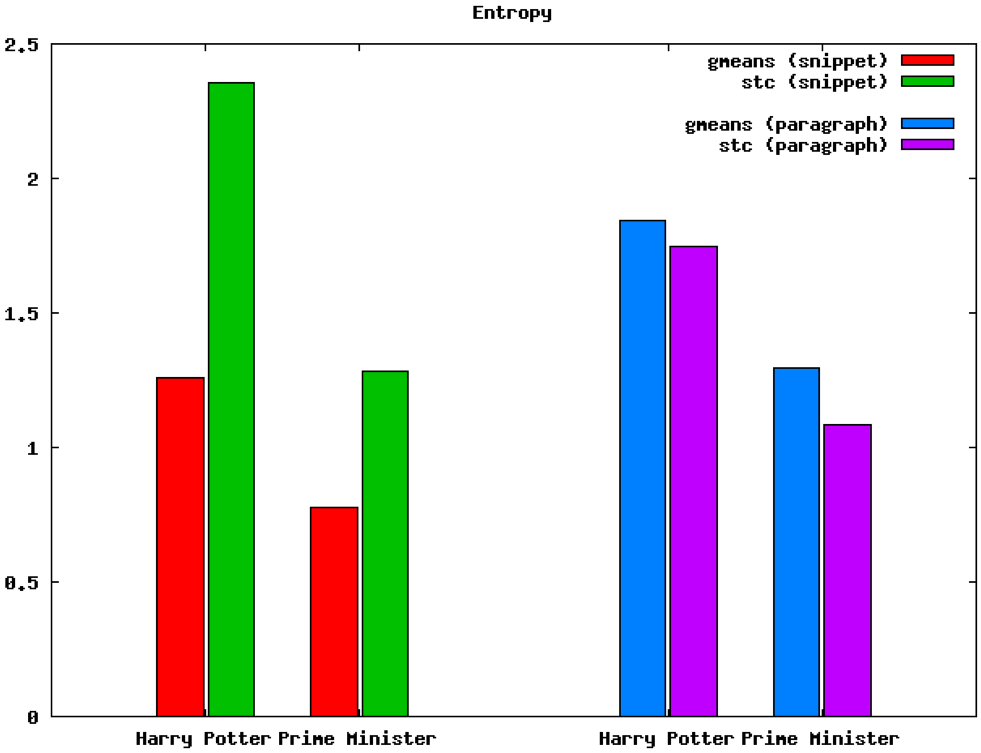


Table 10.4: Generated cluster labels with common words removed.

Intelligence (common words removed)
dept state chief
artificial
agency
nuclear weapon
weapon
emotional
handbook latin american
site route plan
library congress december
challenge read more
competitive
gunnar anzinger
weapon mass destruction
smiley central nhl
napolitano red eye
bin lade
universe
and artificial
and sarai mitnick
library congress january
that quot iran
december iaea director
world factbook
interrogation war congress
our wide range

Table 10.5: Generated cluster labels without common words removed.

Intelligence (common words kept)
artificial intelligence
intelligence agency
emotional intelligence
nuclear weapon
competitive intelligence
and artificial intelligence
intelligence estimate
weapon
extraterrestrial intelligence
handbook latin american
site route plan
library congress december
divine intelligence
read more help
defense intelligence
republic central intelligence
gunnar anzinger
the field artificial
weapon mass destruction
central intelligence agency
smiley central nhl
the follow map
napolitano red eye
intelligence laboratory
aka artificial intelligence
universe
library congress january
and sarai mitnick
that quot iran
world factbook
december iaea director
interrogation war congress
our wide range

Chapter 11

Speed Measure

The clustering algorithms were also compared in speed. 6 collections of different sizes, were generated by random chosen documents from the query ”Computer”. Document size, number of words contained, and the number of baseclusters generated for each collection, can be seen below in table 11.1. The tables show only the values for documents processed with the *paragraph* option. The observant reader might notice that the table only show 5 collections. The 6th was of size 15000, and had to be removed from the paragraph set (it is retained in the snippet set) because the test-computer ran out of memory when running STC.

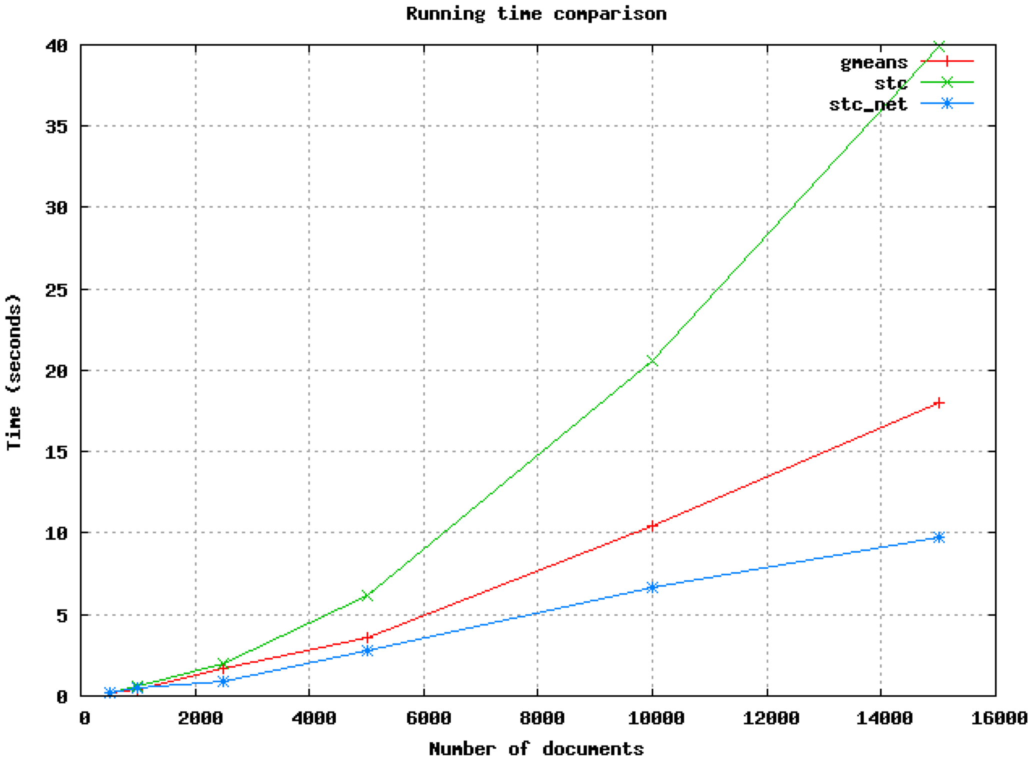
Benchmarking was done using `gmeans`, STC and the distributed version of STC (from now on referred to as `stc_net`). The main computer used was a dual processor, dual core intel cpu of 2.13GHz each, with 3GB RAM running on the linux operating system. `stc_net` was run on four nodes. Two of the nodes were run on the main computer itself, to exploit both it’s processors, the other two were run on two computers, both with the same specifications as the main computer, but with only one processor each.

Figures 11.1 and 12.1 shows the time used by each program, measured in seconds. All programs are theoretically time linear. However, none of them appear completely linear. This is especially evident in STC. Reasons might

Table 11.1: Number of documents and words

Documents	**Words**	**Baseclusters**
10000	6689940	7904383
5000	3228589	3814591
2500	1549639	1833148
1000	636529	758747
500	286993	341772

Figure 11.1: Total running time based on document set size. With snippets.



be related to memory caching or a high algorithmically constant. We should also consider the possibility that there are elements in STC which doesn't run in constant time.

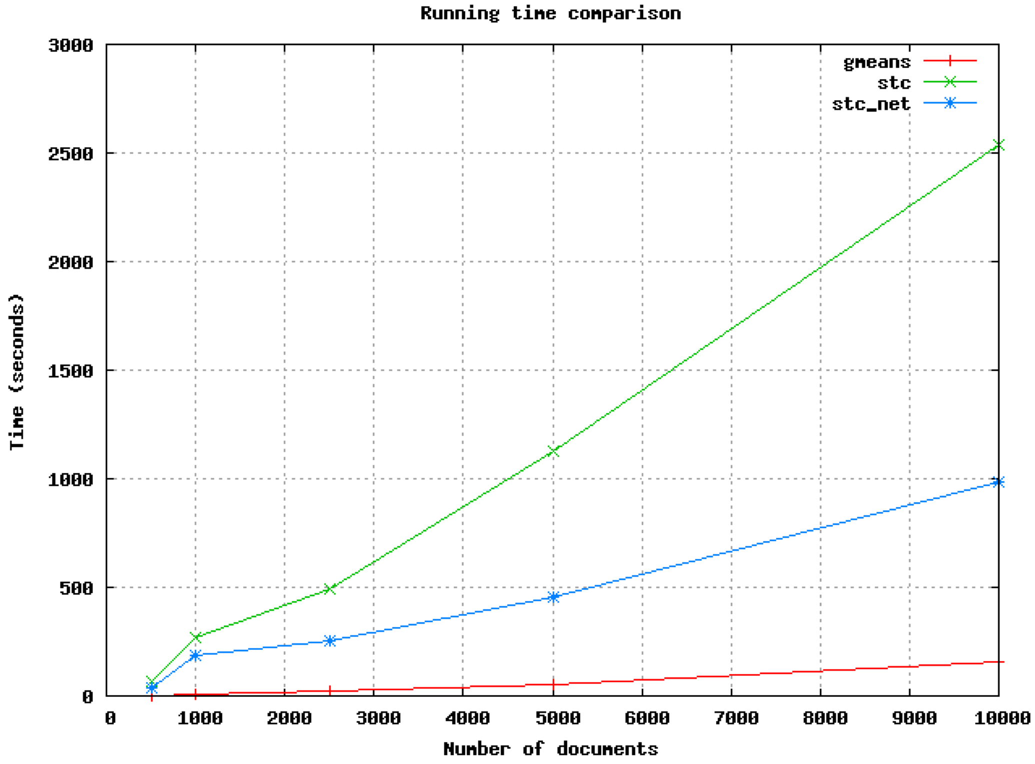
11.0.3 stc_net

As can be seen, `stc_net` using the hardware described, performs at approximately $\frac{2}{5}$ of the speed of STC running on only one processor on one computer. Substantially gain in speed can therefore be gained by running STC on several computers.

11.0.4 gmeans

`gmeans` uses slightly less time without the ”-w -1”-option. But then it would for several document sets run into a loop, and never terminate. This happens when `gmeans` get empty clusters after clustering. The ”-w -1”-option prevents this by trying to reduce the number of clusters. However, at one occation it

Figure 11.2: Total running time based on document set size. With paragraphs.



ran into a loop even with this option enabled. This shows that `gmeans` in it's original form, can't be trusted to always return a set of clusters[1].

[1] Hopefully it might be possible to build a workaround by altering the `gmeans` source code

Chapter 12

Further Discussion and Conclusion

12.1 STC and Number of Base Clusters

The original article [Zamir and Etzioni, 1998] suggests using $k = 500$ base clusters for the final merging. [Stefanowski and Weiss, 2003] uses a predefined threshold, in which all base clusters scoring above the threshold are included in the final merging. They further state: *"We observed that the choice of base cluster score threshold of the STC algorithm is a crucial issue as it strongly affects the number of discovered clusters."*

We tested both methods, and found that even if the number of clusters varies dramatically, the top clusters don't. While threshold values have to be changed according to the size of the document set, $k = 500$ always gives a useful amount of base clusters. On the other hand, an optimal threshold value for one document set, might be inadequate for another. For example when one is larger than the other.

Table 12.1 shows the top ten clusters on the query "intelligence". Figure 12.1 shows the similarity values for threshold values ranging from 10 to 35, using all clusters and only top ten clusters. Number of clusters generated for each threshold can be seen in table 12.2.

As can be seen, the top ten clusters gives a constant value until the threshold produces less than 500 base clusters at the end. The reason similarity values when all clusters are included is much higher, is because many small clusters tend to have high similarity. In either way, only the top clusters are relevant for the user.

Since we have showed that top clusters doesn't vary much, a constant value k is the better candidate. The threshold value score is an unecessary

Table 12.1: Top clusters for two different threshold values.

Threshold value=10	**Threshold value=35**
artificial intelligence	artificial intelligence
intelligence agency	intelligence agency
emotional intelligence	emotional intelligence
nuclear weapon	nuclear weapon
competitive intelligence	competitive intelligence
and artificial intelligence	and artificial intelligence
intelligence estimate	intelligence estimate
the central intelligence	weapon
weapon	extraterrestrial intelligence
intelligence agency the	handbook latin american

Table 12.2: Threshold and number of clusters.

threshold	**clusters**
10	210
15	136
20	110
25	70
30	36
35	27

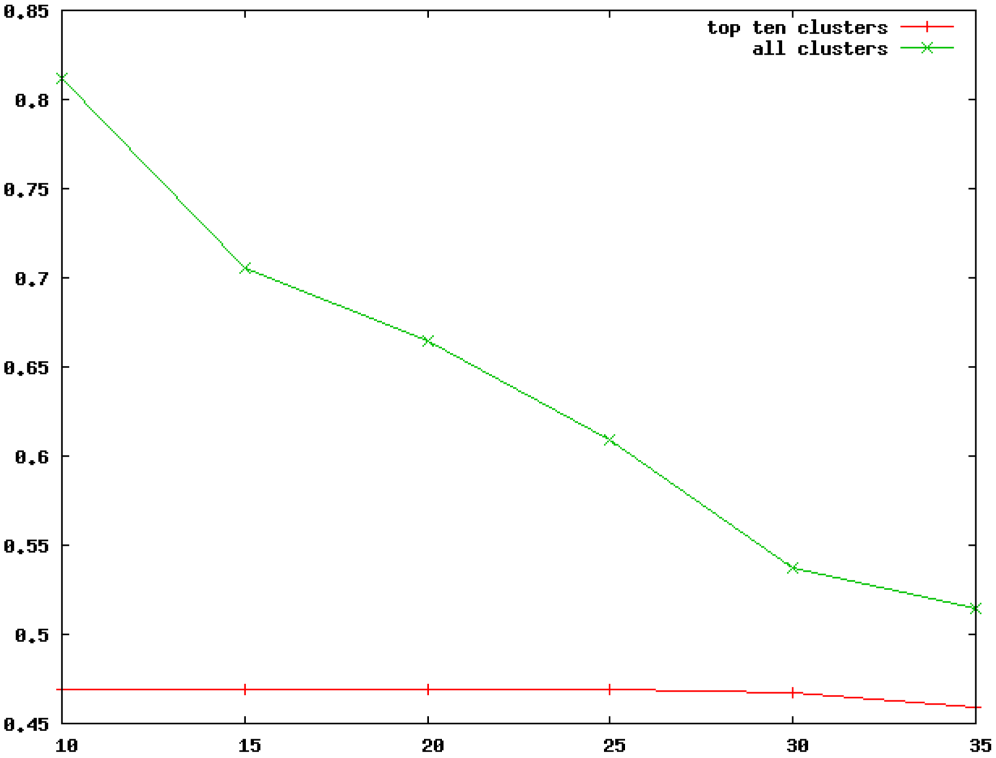
variable, and can be removed from the algorithm completely.

12.2 STC and Hierarchical Clusters

This is best described by an example. For a search on planets in the solar system, ”jupiter saturn” and ”saturn jupiter” will be clustered into different clusters. Since ”jupiter” and ”saturn” also are base clusters, a solution will be to merge the former baseclusters with the latter. However, a base cluster like ”jupiter” will be much larger than ”jupiter saturn”. This might be prevented by adding a rule when merging base clusters: all base clusters contained fully in another, should be merged.

But this is not always wanted. For instance might ”tony blair”, ”gordon brown”, etc. be merged into a base cluster ”prime minister”. The problem might also be fixed by stating that phrases which are palindromes of each other is equal, but this is not compatible with the STC per definition. What we want is some hierarchical form of clustering (not flat). Humans also tends to cluster in overlapping hierarchies, according to [Macskassy et al., 1998].

Figure 12.1: Similarity values with different thresholds.



12.3 Conclusion

We compared the Suffix Tree Clustering algorithm with Elliptic K-Means. We proved that the latter is superior in speed, while the former performs generally better in quality. When using snippets only, STC are only slightly slower than elliptic K-Means for document sets up to 1000 documents in our experiments.

We also created a distributed version of STC. We showed that a substantial speed increase can be gained by running the algorithms on several computers. `stc_net` is never outperformed by `gmeans` in our experiments with snippets, only in the experiments with text paragraphs (full document filtered by a heurestic).

The algorithms were tested by varying the input data, stemming and stopwords were tested with nothing substantial to report. The biggest differences were shown when varying the size of the input data. A snippet of 320 characters gave good results with STC, while giving the full textual content with some simple heurestic employed to filter out garbage data, made Elliptic K-Means perform slightly better.

Clustering algorithms were tested on real web-data. Other reported tests are usually performed on top ranked (good quality) documents, or pure content document sets, such as TREC. STC has been reported to perform ”good enough” with snippets only. We show that on real web-data, STC actually gives better quality using snippets only, and is more than ”good enough”.

As it is not necessary to cluster more than, perhaps, the 500 best ranked documents in a modern search engine, we conclude that STC would be the better choice over Elliptic K-Means. It performs adequately in speed, and better in terms of quality.

STC was also compared with varying base cluster score thresholds, and the constant number of $k = 500$ top base clusters. Although the threshold has a great impact on the number of clusters generated, the top clusters vary little. We conclude that using a constant number of base clusters for base cluster merging, is more useful than using a defined threshold. The threshold value score is an unecessary variable, and can be removed from the algorithm completely.

There is still room for improvement, however. We adressed STC's inability to cope with hierarchical structures. Clusters produced are far from optimal in terms of quality. But document clustering is an exciting field, and we hope to see more improvements in the future.

Appendix A

Programs

build

Input A list of documents and their paths.

Output `dictionary`, `ii`, `id`

Reads and parses each document. All tokens (words or numbers) are extracted, and converted to lowercase. No stemming is performed at this stage. Builds an inverted index. Format of the output files are described in Table 6.1.

makeidf

Input Stemmer, `dictionary`, `ii`

Output `idf`

Traverses all words in the dictionary, and calculates the *idf* score for each term based on its document count. The score values are then saved for later efficient lookups.

stoplist

Input n, `dictionary`, `ii`

Output A list of all words that occur in more than n documents.

Iterates the inverted index, and prints all words occuring in more than the given number of documents.

search

Input Query q, `dictionary`, `ii`

Output A list of all documents matching query q.

Parses the query q. Looks up all keywords in the dictionary, and merges the corresponding lists of documents. Returns the list of documents.

process_doc

Input Format flags, list of documents, query q, `id`, `idf`

Output Retrieved documents, either as sparse matrix or list of terms.

Retrieves and processes all documents based on the given flags. Described in more detail in Section 6.6.

showclusters

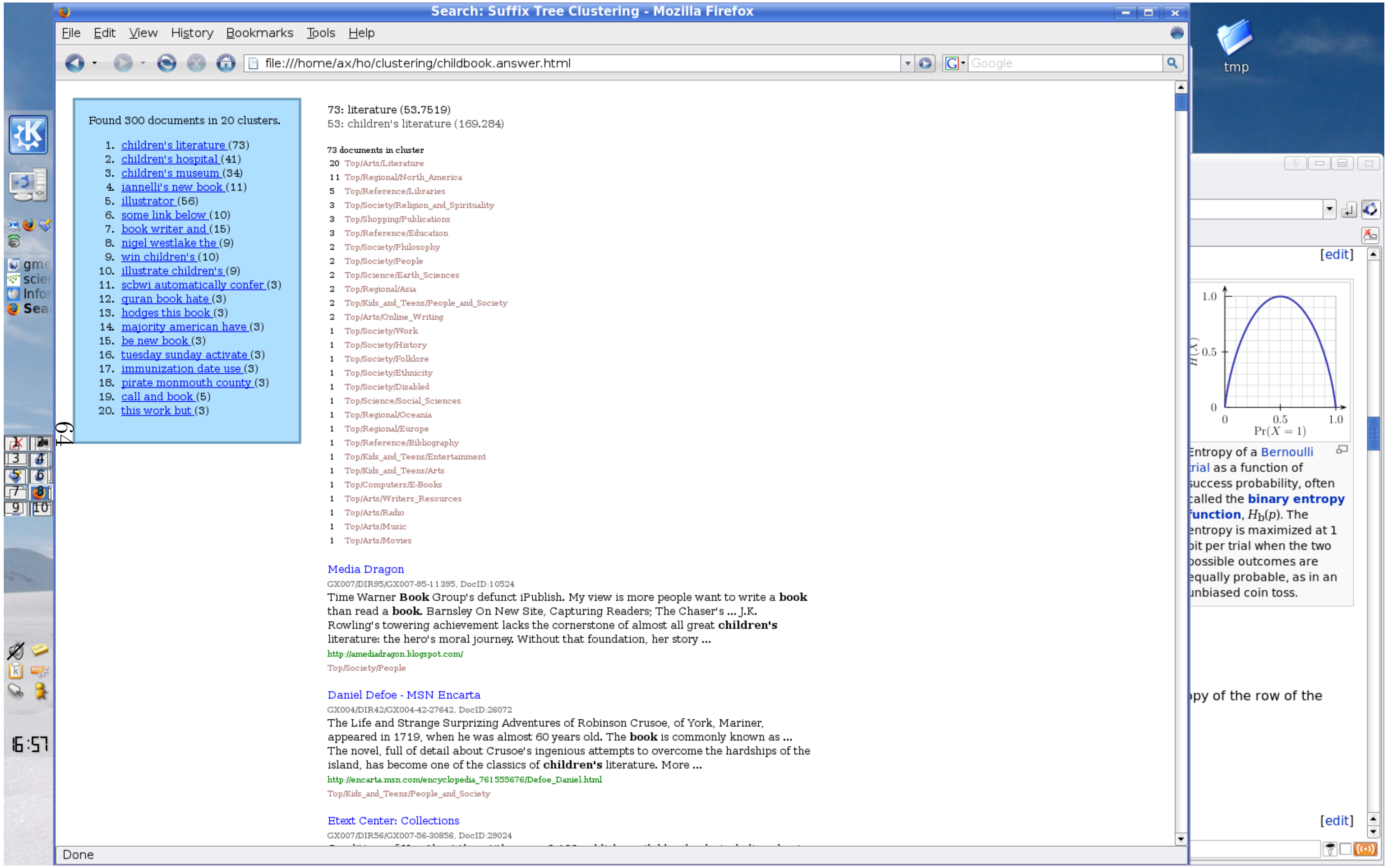
Input List of documents and corresponding clusters, query q, `id`

Output Snippets of all documents sorted in clusters, written to a HTML-file.

Visualizes clusters and documents for easy browsing.

Appendix B

Screenshots of Browser Representation


Search: Suffix Tree Clustering - Mozilla Firefox
File
Edit
View
History
Bookmarks
Tools
Help
file:///home/ax/ho/clustering/childbook.answer.html
Google
tmp
[edit]
1.0
0.5
0
0
0.5
1.0
$H(X)$
$\Pr(X = 1)$
[edit]
Done
6:57

Found 300 documents in 20 clusters.

1. children's literature (73)
2. children's hospital (41)
3. children's museum (34)
4. iannelli's new book (11)
5. illustrator (56)
6. some link below (10)
7. book writer and (15)
8. nigel westlake the (9)
9. win children's (10)
10. illustrate children's (9)
11. scbwi automatically confer (3)
12. quran book hate (3)
13. hodges this book (3)
14. majority american have (3)
15. be new book (3)
16. tuesday sunday activate (3)
17. immunization date use (3)
18. pirate monmouth county (3)
19. call and book (5)
20. this work but (3)

73: literature (53.7519)
53: children's literature (169.284)

73 documents in cluster

20	Top/Arts/Literature
11	Top/Regional/North_America
5	Top/Reference/Libraries
3	Top/Society/Religion_and_Spirituality
3	Top/Shopping/Publications
3	Top/Reference/Education
2	Top/Society/Philosophy
2	Top/Society/People
2	Top/Science/Earth_Sciences
2	Top/Regional/Asia
2	Top/Kids_and_Teens/People_and_Society
2	Top/Arts/Online_Writing
1	Top/Society/Work
1	Top/Society/History
1	Top/Society/Folklore
1	Top/Society/Ethnicity
1	Top/Society/Disabled
1	Top/Science/Social_Sciences
1	Top/Regional/Oceania
1	Top/Regional/Europe
1	Top/Reference/Bibliography
1	Top/Kids_and_Teens/Entertainment
1	Top/Kids_and_Teens/Arts
1	Top/Computers/E-Books
1	Top/Arts/Writers_Resources
1	Top/Arts/Radio
1	Top/Arts/Music
1	Top/Arts/Movies

Media Dragon
GX007/DIR95/GX007-95-11395, DocID:10524
Time Warner **Book** Group's defunct iPublish. My view is more people want to write a **book** than read a **book.** Barnsley On New Site, Capturing Readers; The Chaser's ... J.K. Rowling's towering achievement lacks the cornerstone of almost all great **children's** literature: the hero's moral journey. Without that foundation, her story ...
http://amediadragon.blogspot.com/
Top/Society/People

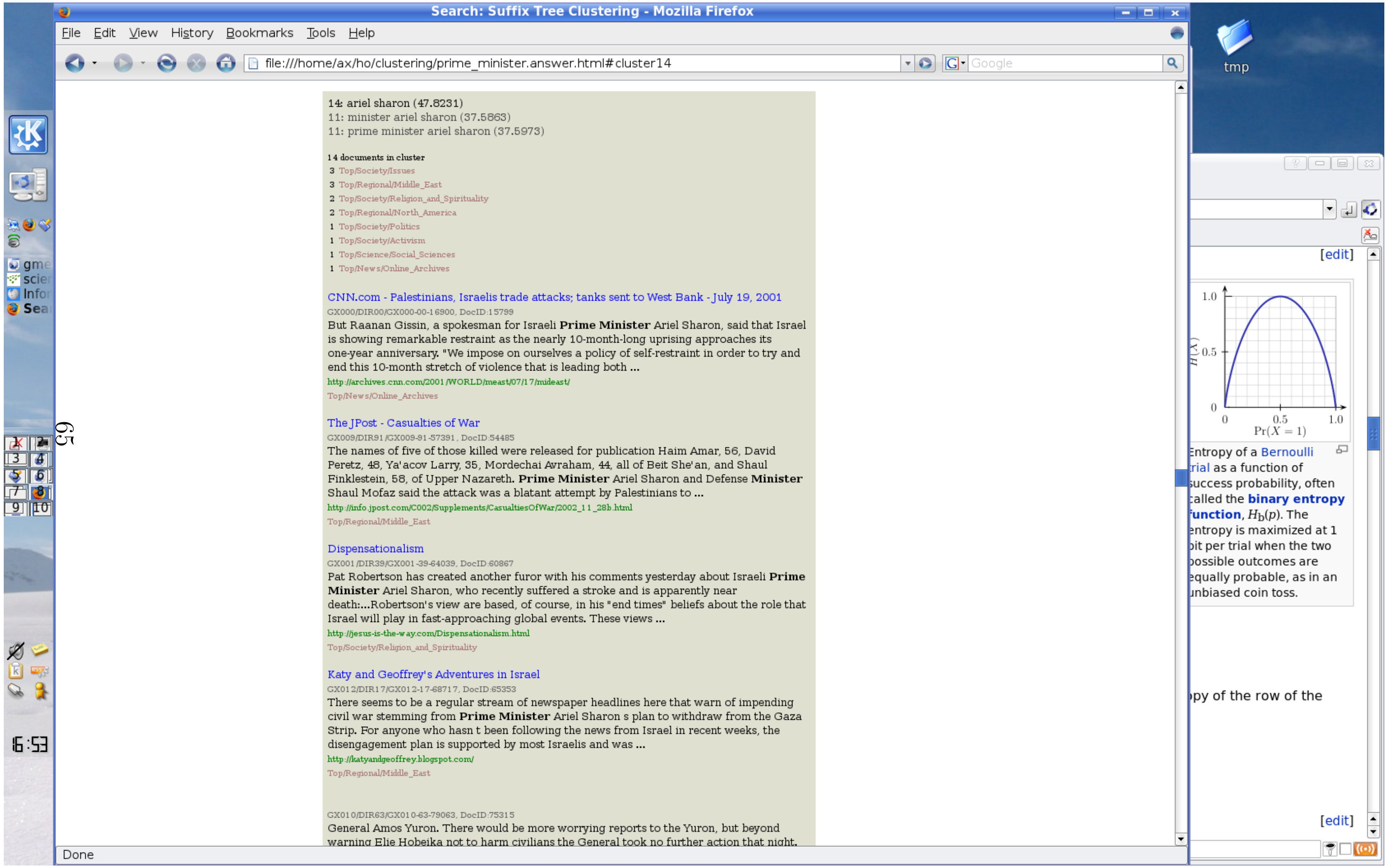
Daniel Defoe - MSN Encarta
GX004/DIR42/GX004-42-27642, DocID:26072
The Life and Strange Surprizing Adventures of Robinson Crusoe, of York, Mariner, appeared in 1719, when he was almost 60 years old. The **book** is commonly known as ... The novel, full of detail about Crusoe's ingenious attempts to overcome the hardships of the island, has become one of the classics of **children's** literature. More ...
http://encarta.msn.com/encyclopedia_761555676/Defoe_Daniel.html
Top/Kids_and_Teens/People_and_Society

Etext Center: Collections
GX007/DIR56/GX007-56-30856, DocID:29024

Entropy of a Bernoulli trial as a function of success probability, often called the **binary entropy function**, $H_b(p)$. The entropy is maximized at 1 bit per trial when the two possible outcomes are equally probable, as in an unbiased coin toss.

py of the row of the

Figure B.1: Snapshot from the browser view. Clusters are listed in the menu.



Search: Suffix Tree Clustering - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///home/ax/ho/clustering/prime_minister.answer.html#cluster14
Google
tmp
Done

14: ariel sharon (47.8231)
11: minister ariel sharon (37.5863)
11: prime minister ariel sharon (37.5973)

14 documents in cluster

3	Top/Society/Issues
3	Top/Regional/Middle_East
2	Top/Society/Religion_and_Spirituality
2	Top/Regional/North_America
1	Top/Society/Politics
1	Top/Society/Activism
1	Top/Science/Social_Sciences
1	Top/News/Online_Archives

CNN.com - Palestinians, Israelis trade attacks; tanks sent to West Bank - July 19, 2001
GX000/DIR00/GX000-00-16900, DocID:15799
But Raanan Gissin, a spokesman for Israeli **Prime Minister** Ariel Sharon, said that Israel is showing remarkable restraint as the nearly 10-month-long uprising approaches its one-year anniversary. "We impose on ourselves a policy of self-restraint in order to try and end this 10-month stretch of violence that is leading both ...
http://archives.cnn.com/2001/WORLD/meast/07/17/mideast/
Top/News/Online_Archives

The JPost - Casualties of War
GX009/DIR91/GX009-91-57391, DocID:54485
The names of five of those killed were released for publication Haim Amar, 56, David Peretz, 48, Ya'acov Larry, 35, Mordechai Avraham, 44, all of Beit She'an, and Shaul Finklestein, 58, of Upper Nazareth. **Prime Minister** Ariel Sharon and Defense **Minister** Shaul Mofaz said the attack was a blatant attempt by Palestinians to ...
http://info.jpost.com/C002/Supplements/CasualtiesOfWar/2002_11_28b.html
Top/Regional/Middle_East

Dispensationalism
GX001/DIR39/GX001-39-64039, DocID:60867
Pat Robertson has created another furor with his comments yesterday about Israeli **Prime Minister** Ariel Sharon, who recently suffered a stroke and is apparently near death:...Robertson's view are based, of course, in his "end times" beliefs about the role that Israel will play in fast-approaching global events. These views ...
http://jesus-is-the-way.com/Dispensationalism.html
Top/Society/Religion_and_Spirituality

Katy and Geoffrey's Adventures in Israel
GX012/DIR17/GX012-17-68717, DocID:65353
There seems to be a regular stream of newspaper headlines here that warn of impending civil war stemming from **Prime Minister** Ariel Sharon s plan to withdraw from the Gaza Strip. For anyone who hasn t been following the news from Israel in recent weeks, the disengagement plan is supported by most Israelis and was ...
http://katyandgeoffrey.blogspot.com/
Top/Regional/Middle_East

GX010/DIR63/GX010-63-79063, DocID:75315
General Amos Yuron. There would be more worrying reports to the Yuron, but beyond warning Elie Hobeika not to harm civilians the General took no further action that night.

[edit]

Entropy of a Bernoulli
rial as a function of
uccess probability, often
alled the **binary entropy**
unction, $H_b(p)$. The
ntropy is maximized at 1
it per trial when the two
ossible outcomes are
qually probable, as in an
nbiased coin toss.

py of the row of the

[edit]

Figure B.2: Snapshot from the browser view of cluster #14.

Bibliography

[Baeza-Yates et al., 1999] Baeza-Yates, R., Ribeiro-Neto, B., et al. (1999). *Modern information retrieval*. Addison-Wesley Harlow, England.

[Bille, 2005] Bille, P. (2005). Suffix trees and applications.

[Broffit et al., 1966] Broffit, J., Morgan, H., and Soden, J. (1966). On Some Clustering Techniques for Information Retrieval. *Report ISR*, 1(11).

[Bush, 1996] Bush, V. (1996). As we may think. *interactions*, 3(2):35–46.

[Chakrabarti, 2003] Chakrabarti, S. (2003). *Mining the Web: Discovering Knowledge from Hypertext Data*. Morgan Kaufmann.

[Chudnovsky, 2005] Chudnovsky, A. (2005). Indexing good content – not junk.

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

[Dhillon et al., 2001] Dhillon, I. S., Fan, J., and Guan, Y. (2001). Efficient clustering of very large document collections. In R. Grossman, C. Kamath, V. K. and Namburu, R., editors, *Data Mining for Scientific and Engineering Applications*, pages 357–381. Kluwer Academic Publishers. Invited book chapter.

[Dhillon and Modha, 2001] Dhillon, I. S. and Modha, D. S. (2001). Concept decompositions for large sparse text data using clustering. *Machine Learning*, 42(1):143–175.

[Dom, 2001] Dom, B. (2001). An information-theoretic external cluster-validity measure. *Research Report RJ*, 10219.

[Fellbaum et al., 1998] Fellbaum, C. et al. (1998). *WordNet: an electronic lexical database.* Cambridge, Mass: MIT Press.

[Gusfield, 1997] Gusfield, D. (1997). *Algorithms on strings, trees, and sequences.* Cambridge University Press New York.

[Harman, 1993] Harman, D. (1993). Overview of the first TREC conference. *Proceedings of the 16th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 36–47.

[Hearst and Pedersen, 1996] Hearst, M. A. and Pedersen, J. O. (1996). Re-examining the cluster hypothesis: scatter/gather on retrieval results. In *SIGIR '96: Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 76–84, New York, NY, USA. ACM.

[Jardine and van Rijsbergen, 1971] Jardine, N. and van Rijsbergen, C. (1971). The Use of Hierarchic Clustering in Information Retrieval. *Information Storage and Retrieval*, 7(5):217–240.

[Lewis, 1995] Lewis, P. H. (1995). Digital equipment offers web browsers its 'super spider'. *The New York Times.*

[Luhn, 1957] Luhn, H. (1957). A statistical approach to mechanized encoding and searching of literary information. *IBM Journal of Research and Development*, 1(4):309–317.

[MacQueen, 1967] MacQueen, J. (1967). Some methods for classification and analysis of multivariate observations. *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, 1(281-297):14.

[Macskassy et al., 1998] Macskassy, S., Banerjee, A., Davison, B., and Hirsh, H. (1998). Human Performance on Clustering Web Pages. *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM Press, New York*, pages 264–268.

[Mitchell, 1997] Mitchell, T. (1997). *Machine Learning.* McGraw-Hill.

[Nelson, 1996] Nelson, M. (1996). Fast String Searching With Suffix Trees. *Dr. Dobbś Journal*, pages 115–119.

[Porter, 1980] Porter, M. (1980). An Algorithm for Suffix Stripping Program. *Program*, 14(3):130–137.

[Raggett et al., 1999] Raggett, D., Le Hors, A., and Jacobs, I. (1999). HTML 4.01 Specification. *W3C Recommendation REC-html401-19991224, World Wide Web Consortium (W3C), Dec.*

[Raghavan and Wong, 1986] Raghavan, V. and Wong, S. (1986). A critical analysis of vector space model for information retrieval. *Journal of the American Society for Information Science*, 37(5):279–287.

[Rasmussen, 1992] Rasmussen, E. (1992). Clustering algorithms.

[Salton, 1971] Salton, G. (1971). The SMART Retrieval System — Experiments in Automatic Document Processing.

[Salton et al., 1975] Salton, G., Wong, A., and Yang, C. (1975). A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620.

[Singhal, 2001] Singhal, A. (2001). Modern Information Retrieval: A Brief Overview. *Bulletin of the Technical Committee on Data Engineering.*

[Stefanowski and Weiss, 2003] Stefanowski, J. and Weiss, D. (2003). Carrot2 and language properties in web search results clustering. *Proceedings of the International Atlantic Web Intelligence Conference.*

[Steinbach et al., 2000] Steinbach, M., Karypis, G., and Kumar, V. (2000). A comparison of document clustering techniques. *KDD Workshop on Text Mining*, 34:35.

[Strehl et al., 2000] Strehl, A., Ghosh, J., and Mooney, R. (2000). Impact of similarity measures on web-page clustering. *Proc. AAAI Workshop on AI for Web Search (AAAI 2000), Austin*, pages 58–64.

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

[Van Rijsbergen, 1979] Van Rijsbergen, C. (1979). *Information Retrieval.* Butterworth-Heinemann Newton, MA, USA.

[Weiner, 1973] Weiner, P. (1973). Linear pattern matching algorithms. *Proceedings of the 14th IEEE Symposium on Switching and Automata Theory*, pages 1–11.

[Willett, 1988] Willett, P. (1988). Recent trends in hierarchic document clustering: a critical review. *Information Processing and Management: an International Journal*, 24(5):577–597.

[Zamir and Etzioni, 1998] Zamir, O. and Etzioni, O. (1998). Web document clustering: a feasibility demonstration. In *SIGIR '98: Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 46–54, New York, NY, USA. ACM.

[Zamir et al., 1997] Zamir, O., Etzioni, O., Madani, O., and Karp, R. (1997). Fast and intuitive clustering of web documents.

[Zhong, 2005] Zhong, S. (2005). Efficient online spherical K-means clustering. *Neural Networks, 2005. IJCNN'05. Proceedings. 2005 IEEE International Joint Conference on*, 5.