NTNU
Innovation and Creativity

On-Line Clustering of Web Search Results

Hans Olaf Borch

Master of Science in Computer Science
Submission date: July 2006
Supervisor: Jon Atle Gulla, IDI

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

Problem Description

This project will develop a prototype for on-line clustering of web search results. The state-of-the-art for document clustering systems will be surveyed, and the clustering algorithm that seems most promising will be used in the prototype. The grouping of documents and the quality of the cluster labels will be assessed through an evaluation. In addition to discussing various techniques for achieving good label quality, the project will investigate the use of additional knowledge bases for improving the label quality. This includes the ODP category hierarchy and a query log from a web search engine.

Assignment given: 20. January 2006
Supervisor: Jon Atle Gulla, IDI

Preface

This report presents the master thesis of my 5^{th} year in the "Sivilingeniør i Datateknikk" course at NTNU, Trondheim. The work was carried out at the Department of Computer and Information Science, Faculty of Information Technology, Mathematics and Electrical Engineering at NTNU under supervision of Professor Jon Atle Gulla.

I would first of all like to thank Professor Jon Atle Gulla for his co-operation. He has given me great freedom in selecting the topic for my thesis and given me helpful feedback throughout the semester.

My assisting supervisor Jon Espen Ingvaldsen has contributed with ideas during the semester and has been helpful in reviewing the report. I would also like to thank Per Gunnar Auran from Yahoo who has contributed with ideas as well as providing me with insides into the world of commercial search engines.

Finally, I would like to thank master degree candidates Tarjei Lægreid, Paul Christian Sandal and Geir Øyvin Grimnes for their assistance in evaluating the prototype presented in this report.

Trondheim, July 2, 2006,

Hans Olaf Borch

Contents

Part I

Introduction

CHAPTER 1
INTRODUCTION

1.1 Background

As the Internet grows, web search engines are continuously improving and can now handle billions of documents. In result, a typical user query results in millions of hits. Overwhelmed by the number of results, many users investigate only the two or three top ranked documents. If nothing interesting is found most users reformulate their query rather than sift through many pages of search results. A large study done using a query log from the AltaVista search engines revealed that users tend to type short queries and only look at the first results [7].

A way of assisting users in finding what they are looking for quickly is to group the search results by topic. The user does not have to reformulate the query, but can merely click on the topic most accurately describing his or her spesific information need. The process of grouping documents is called *clustering* and was originally used for analysing numerical data in the field of data mining. It has since been adapted to suit the needs for clustering textual documents.

The two main challenges with adapting clustering to suit the needs of web search engines and textual data has been to give good descriptive labels to the clusters and to be able to cluster documents on-the-fly in response to a particular user query. Traditional data mining approaches are not concerned with labelling clusters, but in return they are often very good at grouping data. Unfortunately, regardless of how good the document grouping is, users are not likely to use a clustering engine if the labels are poor. Clustering performance is also a major issue, because web users expect fast response times. To deal with this linear time clustering algorithms that can cluster hundreds of documents in under a second have been developed.

Several commercial clustering engines exist. The most famous is the Vivisimo engine, which was rewarded by SearchEngineWatch.com for best meta-search engine from 2001 to 2003. Large companies such as Google and Microsoft also seem to be interested in clustering and the technology has been called "the PageRank of the future" [23, 27].

1.2 Approach

As described in the problem statement, this project will develop a prototype for on-line clustering of web search results. A clustering engine relies on a search engine to give a ranked list of matches releavant to a given query. Therefore the first step is to develop a search engine using standard information retrieval techniques. The the clustering is built on top of this, using an algorithm called Suffix Tree Clustering (STC). A technique for labelling clusters based on the clustering algorithm will also be developed. Finally, a web interface will be developed to allow users easy access to the system.

1.3 Contributions

Most articles available on clustering describe a clustering algorithm and how it is used to group documents, but few articles focus on the actual labelling of the clusters. This is naturally a crucial part for the end-user, who might fail to see the added value of clustering the results if the labels are bad. In addition, the articles that actually describe the process of labelling the clusters often leave out details or contain certain unspecified elements.

The main contribution of this thesis is to show how a clustering engine complete with a search engine and state-of-the-art post-retrieval clustering mechanism can be developed. It builds on the well-known Suffix Tree Clustering algorithm, and through thorough testing of the prototype most of the issues not discussed in the original article on STC are addressed.

In addition, experimenting with the prototype sheds new light on many of the choices made by the authors of the clustering algorithm, including the use of stemming, removal of stopwords, join criteria for clusters and most importantly on labelling. The possibilities of using additional knowledge bases for improving cluster labels are also explored.

1.4 Document Outline

The document is organized in four parts. Part I is this introductory chapter. Part II presents the theoretical background, including an overview of the technology used in later chapters and a state-of-the-art survey. Part III describes the approach taken to document clustering and the implementation of the prototype system. Part IV presents the evaluation of the prototype, along with a discussion of the results and a concluding chapter.

Part II

Theoretical Background

CHAPTER 2
TECHNOLOGICAL OVERVIEW

Techniques drawn from information retrieval, data mining and text mining are used throughout the thesis. This chapter gives background information and discusses the various techniques and concepts used in later chapters.

2.1 Classical Information Retrieval

In essence, an information retrieval system processes a *document collection* to select a set of *index terms* and build an *index* that allows searching the documents efficiently. Given a *model* for representing documents, the query can be matched against each document in the collection and a list of relevant documents is then returned to the user.

The simplest strategy for selecting index terms is to use *all* words in all documents. Additional strategies are aimed at either reducing *index size* or improving *retrieval performance*. Strategies include removing stop words and stemming or lemmatizing the words before indexing.

Having selected the index terms, different models exists for representing the documents. The classic models are the Boolean model, the vector space model and the probabilistic models. Only the *vector space model* is presented here since it is the most widely adopted and the one used in the prototype described later.

In the model, each document is represented in the term space corresponding to the union of all index terms appearing in the document collection [4]. A document collection of n documents $D = \{d_1, ..., d_n\}$ containing m unique words can be described by means of n document vectors $d_j = (w_{j1}, ..., w_{jm})$, where w_{ji} designates a weight for the term t_i in document d_j. The most widely adopted scheme for term weighting called $tf * idf$ is defined as

$$w_{i,j} = tf * idf = \frac{freq_{i,j}}{max_l \, freq_{l,j}} * \log \frac{N}{n_i} \tag{2.1}$$

where $freq_{i,j}$ is the raw frequency of t_i in document d_j and the maximum frequency is computed over all words occurring in d_j. N is the number of documents in the collection and n_i is the number of documents containing the term t_i. The $tf * idf$-measure tries to balance two effects: Words frequently occurring in a document are likely to be important (hence the tf or *term frequency* part), but words occurring in a large percentage of the document collection are less discriminating and thus less important for retrieval (hence the idf or *inverse document frequency* part). The denominator of the term frequency is a way of normalizing the term frequency according to the length of the document, so that long documents are not favoured.

Although several alternatives have been proposed, some variant of $tf * idf$ is often used for weighting. Several variations are described in a paper by Salton and Buckley [19]. For weighting query terms they suggest:

$$w_{i,q} = \left(0.5 + \frac{0.5\ freq_{i,q}}{max_l\ freq_{l,q}}\right) * \log\frac{N}{n_i} \tag{2.2}$$

where w_{iq} is the weight for the term t_i in the query q, N is the number of documents in the collection and n_i is the number of documents containing the term t_i. The similarity between two term vectors is determined by measuring their vector distance. A common measure is the *cosine-similarity* φ defined as:

$$\varphi(d_i, d_j) = \frac{\vec{d_i} \bullet \vec{d_j}}{|\vec{d_i}| * |\vec{d_j}|} \tag{2.3}$$

where $\vec{d_i}$ is the document vector for document d_i. The vector model is a simple and fast strategy that has the advantage of allowing partial matches, and provides improved retrieval performance by ranking the results according to similarity to a query [4].

2.2 Document Clustering

Clustering is the process of grouping documents with similar contents. Initially, clustering was used in information retrieval systems for increasing precision or recall [21] and finding similar documents. Later it has been used for browsing document collections [8] and for automatically building taxonomies [14]. In this thesis however, we focus on using clustering to organize results from a search engine like in [16].

Although it is possible to cluster documents before the user submits a query, the amount of information and variety of topics present on the Internet suggests that we need to come up with specialized categories in response to each user query. This has been referred to as *on-line* or *query-time* clustering, which obviously introduces major performance requirements. A typical web user is not very patient, and will be annoyed by having to wait more than a couple of seconds at most for the results of his request. Therefore the performance of clustering is a major concern. This thesis focuses on the suitability of clustering for on-line grouping of documents returned by a search engine into categories with descriptive labels.

Clustering is traditionally a part of data mining where it is usually applied to numerical data. The document clustering techniques can be divided in the techniques directly adapted from data mining and the algorithms defined specifically with unstructured text in mind.

Document clustering shares several characteristics with the field of *text categorization*. The main difference is that categorization focus on assigning documents to *predefined groups*, where as document clustering tries to *extract* the groupings inherent in the documents. Many text categorization approaches build classifyers that are trained to categorize documents efficiently, but this would not work for clustering since the categories are not know in advance.

Document clustering essentially has two major challenges: to group similar documents into coherent clusters, and to label these clusters with descriptive labels. The former challenge is often addressed by transforming the documents into vectors and then using well tested data mining techniques for clustering numerical data in order to produce the clusters. The labelling is more difficult, especially for the document vector approaches. While clustering has been researched for decades, only in the last decade has specialized clustering algorithms designed for dealing with unstructured textual data appeared.

The next section discusses why we should cluster search results, and then some of the well-known approaches to clustering are described.

2.2.1 Why Cluster Search Results

Search results on the web are traditionally presented as a flat ranked list of documents, frequently millions of documents long. The main use for clustering is not to improve the actual ranking, but to give the user a quick overview of the results. Having divided the result set into clusters, the user can quickly narrow down his search further by selecting a cluster. This resembles *query refinement*, but avoids the need to query the search engine for each step.

Evaluations done using the Grouper system [35] indicate that users tend to investigate more documents per query than in normal search engines. It is assumed that this is because the user clicks on the desired cluster rather than reformulating his query. The evaluation also indicates that once one interesting document has been found, the users often find other interesting documents in the same cluster.

2.2.2 Traditional Clustering Approaches

In general, these techniques transform the documents to vectors, and employs standard means of calculating differences between vectors to cluster the documents.

Hierachical Clustering

In general, there are two types of hierarchical clustering methods [10]:

Agglomerative or bottom-up hierarchical methods create a cluster for each document and then merges the two most similar clusters until just one cluster remains or some termination condition is satisfied. Most hierarchical clustering methods fall into this category, and they only differ in their definition of intercluster similarity and termination conditions.

Divisive or top-down hierarchical methods do the opposite, by starting with all documents in one cluster. The initial cluster is divided until some termination condition is satisfied.

Hierarchical methods are widely adopted, but often struggle to meet the speed requirements of the web. Usually operating on document vectors with a time complexity of $O(n^2)$ or more, clustering more than a few hundred snippets is often unfeasible. Another problem is that if two clusters are incorrectly merged in an early state there is no way of fixing this later in the process. Finding the best halting criterion that works well with all queries can also be very difficult.

K-Means Clustering

The K-Means algorithm comes in many flavours and produces a fixed number (k) of flat clusters. The algorithms generally follow the following process: Random samples from the collection are drawn to serves as *centroids* for initial clusters. Based on document vector similarity, all documents are assigned to the closest centroid. New centroids are calculated for each cluster, and the process is repeated until nothing changes or some termination condition is satisfied.

The process can be speeded up by clustering a subset of the documents, and later assign all documents to the precomputed clusters. Several problems exist with this approach: It can only produce a fixed number of clusters (k). It performs optimally when the clusters are spherical but we have no reason to assume that documents clusters are spherical. Finally, a "bad choice" in the random selection of initial clusters can severely degrade performance.

Buckshot and Fractation Algorithm

These linear time clustering algorithms were both introduced in [8], and are so-called *seed-based partitional* clustering techniques. Partitional clustering follows a three-step process: (1) Find k cluster centers, (2) Assign each document to the nearest cluster, (3) refine the partitioning. Buckshot and Fractation are different strategies for generating the initial k cluster centers from n documents (1). The idea is to employ a slow (typically quadratic) but high-quality clustering techniques on a subset of the data, and then uses the results to approximate the final clusters.

Buckshot selects $\sqrt{kn}$ of the documents and applies the clustering algorithm, and thus runs in $O(nk)$ time. Buckshot is a non-deterministic but fast technique. Fractation splits the document collections into a m buckets ($m > k$) and clusters each bucket. These clusters are then treated as the individuals and the process is repeated until only k clusters remain. This process can be shown to run in $O(mn)$ time.

The rest of the documents are then assigned to their nearest cluster center based on some heuristic. In step three clusters are refined either by re-applying the nearest cluster center approach (resembling the k-means approach), or by splitting and/or joining cluster based on some heuristic of overlap/disjointness.

According to the authors, Buckshot achieves high performance and is thus better suited for *roughly* clustering on-line, while Fractation can be used to determine primary partitioning of an entire corpus. They propose a technique (Scatter/Gather [8]) that uses Fractation to create an initial partitioning and then uses Buckshot to do on-the-fly clustering to tailor the results from the Fractation algorithm towards the specific user request at query-time.

2.2.3 Text-Oriented Clustering Approaches

These techniques are characterized by their focus on words rather than document vectors. Instead of representing documents as vectors the typically focus on grouping documents that share sets of frequently occurring phrases.

Frequent Itemsets Clustering

Wang, Fung and Ester [3] propose using the data mining notion of *frequent itemsets* to cluster documents. Frequent itemsets originate from *association rule mining* typically used in data warehouses. The idea is that documents that share a set of words that appear frequently are related, and this is used cluster documents. The article also presents a way to infer hierarchies of clusters.

Ferragina and Gulli [9] propose a similar method that mines for so-called *gapped sequences*, that do not necessarily appear continuously in the text. They also cluster based on the occurrence of such frequent phrases and build hierachical clusters.

Suffix Tree Clustering

The Suffix Tree Clustering (STC) algorithm was introduced by Zamir and Etzioni [16] and further developed in [34]. The algorithm focuses on clustering snippets faster than standard data mining approaches to clustering by using a data structure called a suffix tree. Its time complexity is linear to the number of snippets, making it attractive when clustering a large number of documents. Since this is the core clustering approach used by the prototype presented in this thesis, the algorithm will be presented in detail.

The algorithm consists of three steps: (1) Document cleaning, (2) Identifying base clusters, and (3) combining base clusters into clusters. These steps are demonstrated by running the algorithm on the following document collection (resembling sample web documents titles):

```
Document 1: Jaguar car reviews - Review Centre
Document 2: ## PANTERA ONCA ##
Document 3: Jaguar reviews!
Document 4: Buy Pantera Onca Pictures
```

1. **Document cleaning:** This step uses a light stemming algorithm on the text. Sentence boundaries are marked and non-word tokens are stripped. After the pre-processing we are left with the following "sentences" for each of the documents:

   ```
   Document 1: {jaguar car review, review centre}
   Document 2: {pantera onca}
   Document 3: {jaguar review}
   Document 4: {buy pantera onca picture}
   ```

2. **Identifying base clusters:** The process of identifying base clusters resembles building an inverted index of phrases for the document collection. The data structure used is a *suffix tree* [32], which can be built in time linear to the collection size [20].

   The following definition of a suffix tree is adapted from [16]: Formally, a suffix tree is a rooted, directed tree. Each internal node has at least 2 children. Each edge is labeled with a non-empty sub-string of S (hence it is a *trie*). The label of a node is defined to be the concatenation of the edge-labels on the path from the root to that node. No two edges out of the same node can have edge-labels that begin with the same word (hence it is *compact*). For each suffix s of S, there exists a *suffix-node* whose label equals s.

   Figure 2.1 illustrates the suffix tree built using the sentences identified in our sample document collection. Each node represents a phrase and has a list of document IDs in which the phrase occurs.

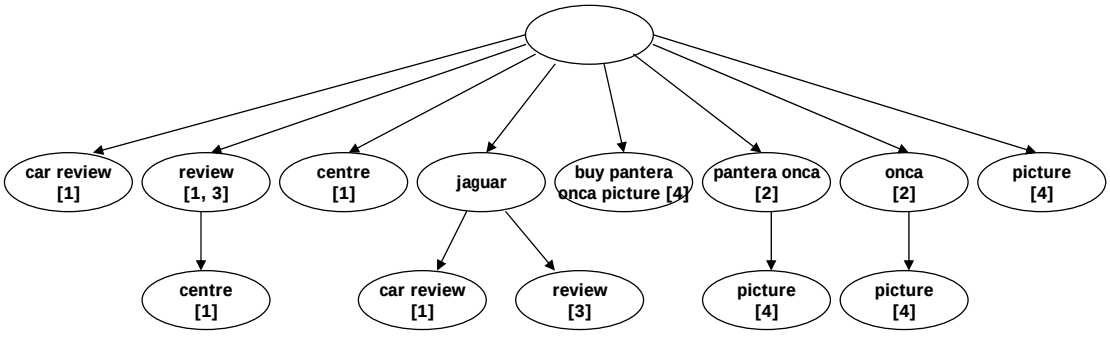

Figure 2.1: Suffix tree of the sample document collection

   Each node in the tree represents a group of documents (a *base cluster*) and is labelled with a phrase that is common to all of them. All groups containing two or more documents are selected to serve as the base clusters.

   Each of the base clusters are assigned a score according to formula 2.4.

Phrase	Documents	Score
review	1, 3	2
jaguar	1, 3	2
pantera onca	2, 4	4
onca	2, 4	2

Table 2.1: Base Clusters

$$s(B) = |B| \cdot f(|P|) \tag{2.4}$$

where $|B|$ is the number of documents in base cluster B, and $|P|$ is the number of words in P that have a non-zero score (the *effective length* of the phrase). Words that appear in more than 40% of the collection or in less than three documents receive a score of zero. The function f penalizes single word phrases, is linear for phrases of two to six words, and is constant for longer phrases.

The scored base clusters are shown in table 2.1. Note that since there are only four documents in the sample collection, no words have been given a zero-score.

3. **Combining base clusters:** This step merges base clusters with highly overlapping document sets. The similarity of base clusters B_n and B_m is a binary function ψ defined as

$$\psi(B_m, B_n) = \begin{cases} 1 & \text{iff } |B_m \cap B_n| \,/\, |B_m| > 0.5 \text{ and } |B_m \cap B_n| \,/\, |B_n| > 0.5 \\ 0 & \text{otherwise} \end{cases} \tag{2.5}$$

where $|B_m \cap B_n|$ is the number of documents shared by B_m and B_n. Calculating this similarity between all base clusters we can create a *base cluster graph*, where nodes are base cluster, and two nodes are connected iff the two base clusters have a similarity of 1. Using this graph a cluster is defined as a connected component in the graph.

The following algorithm is adapted from Cormen et al [6] and presents a simple way of determining the connected components in a graph G, given a function SET(v) that returns the set containing the vertex (base cluster) v.

CONNECTED-COMPONENTS(G)

```
1  for each vertex v ∈ V[G]
2       do make new set containing v
3  for each edge (u, v) ∈ E[G]
4       do if SET(u) ≠ SET(v)
5            then join the sets u and v
```

Figure 2.2 shows the graph with its connected components.

Each connected components constitutes a cluster, and it consists of the union of the documents contained in each of its base clusters. The original article on STC states that "The final clusters are scored and sorted based on the scores of their base clusters and their overlap", without giving any detail as to how this is done. The article says nothing about labelling the clusters either.

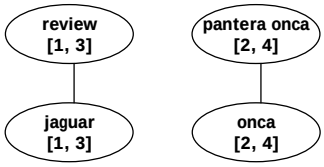

Figure 2.2: Base Cluster graph

2.3 Document Snippets

A common techniqe used by clustering engines is to cluster so-called *document snippets* rather than entire documents. Snippets are the small paragraphs often displayed along with web search results to give the user an indication of the document contents. Figure 2.3 shows the first three results along with their snippets generated by the Google search engine.

Snippets are considerably smaller than the documents (typically only 100-200 characters), thereby drastically reducing the computational cost of the clustering. This is very important since scalability and performance are major challenges for most clustering engines. When building clusters based only on short extracts from the documents, the quality of the snippets returned by the search engine naturally becomes very important. Snippet generation approaches vary from naive (e.g. first words in the document) to more sophisticated (e.g. display the passage containing the most words from the query or multiple passages containing all or most of the query keywords).

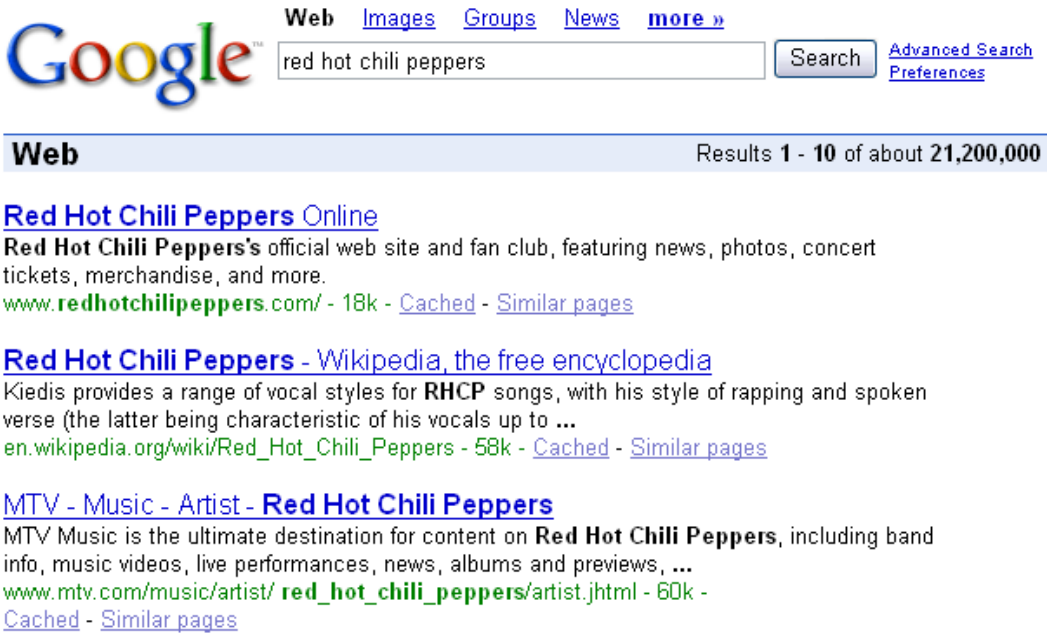

Figure 2.3: Snippets from Google

Clustering algorithms differ in their sensitivity to document length, but generally the effect of using snippets as opposed to entire documents is surprisingly small as demonstrated by [34]. Only about 15% average loss of precision for the clusters was found when using snippets rather than entire documents. The article suggests that this is caused by the search engines

christmas tree
Marcasite Enamel
cat
diamond stud earrings
Golf Water Globe
song lyrics
MD Formulations
shockwave . com
barbie styling
winter jackets
mALL mADNESS GAME BOARD
flatwear
christmas cards
top websites
wwe.com

Table 2.2: Excerpt from a query log

efforts to extract meaningful snippets concerning the user query, which reduces the noise present in the original document so much that the results do not deteriorate significantly. This further emphasises the importance of high quality snippet extraction for snippet clustering approaches.

2.4 Query Log

A query log is a list of queries submitted by users to a search engine. Information stored ranges from only the query to details about each of the resulting pages the user has clicked on, special fields used by the search engine etc. Figure 2.2 shows an example excerpt from a query log, illustrating a simple log file and how diverse user requests can be.

Query logs can contain several types of interesting information: In most cases, it is used to monitor user behaviour in order to either personalize the search engine or to improve the service based on common behavioural patterns. Several large scale studies have investigated query logs from web search engines in order to discover how users normally interact with the engine [1, 7].

Another interesting aspect of the query log is its ability to capture the vocabulary of the users, since users formulate queries according to their perception of a domain, not according to the contents of the documents of the domain.

2.5 Open Directory Project (ODP)

The *Open Directory Project* (ODP) is the most comprehensive human-edited directory of the Web. A global community of volunteer editors maintains this hierarchy of over 700000 categories [28]. It is used in a variety of web applications around the world, and perhaps most famously by Google in their directory service [25].

The information stored in the hierarchy is contained in two RDF-files. The first file is the *content* file that defines each category with a category ID and optional descriptions and links to web pages. The following listing shows a portion of the contents file:

```
<Topic r:id="Top/Computers">
  <tag catid="4"/>
  <d:Title>Computers</d:Title>
  <link r:resource="http://www.cs.tcd.ie/FME/"/>
  <link r:resource="http://pages.whowhere.com/computers/Timeline.html"/>
</Topic>

<ExternalPage about="http://www.cs.tcd.ie/FME/">
  <d:Title>FME HUB</d:Title>
  <d:Description>Formal Methods Europe (FME) is a European organization
  supported by the Commission of the European Union. </d:Description>
</ExternalPage>
```

The second file is the *structure* file that defines the relationships between the categories as demonstrated in the following listing:

```
<Topic r:id="Top/Shopping/Niche/Politics">
  <catid>183341</catid>
  <d:Title>Politics</d:Title>
  <narrow2 r:resource="Top/Shopping/Niche/Politics/Socialism"/>
  <narrow2 r:resource="Top/Shopping/Niche/Politics/Conservatism"/>
  <narrow2 r:resource="Top/Shopping/Niche/Politics/Social_Liberalism"/>
</Topic>
```

CHAPTER 3
STATE-OF-THE-ART SURVEY

The literature offers several solutions to the problem of clustering web documents. Unfortunately, very little information is available about the commercial systems, and the performance of the approaches described in the literature is far from that of the commercial systems. Even though the general approach is described to a point where you could implement a similar system, the tuning and all the little tricks learned through implementing a commercial system are seldom described in the articles. It is likely that they use a somewhat standard clustering algorithm at the base of the approach, but they might also use additional information sources such as lists of named entities or rules etc. to further improve the cluster quality. This chapter presents a way of classifying the approaches and surveys related work on web document clustering.

3.1 Classifying Web Clustering Approaches

We will use the term *web clustering approach* to refer to all aspects of the clustering process for web documents: the data acquisition and processing, the clustering algorithm, the labelling process etc. The following list presents terms and concepts relevant to describing web clustering approaches:

Clustering algorithm: Different clustering algorithms are used at the base of the clustering approach. For details on different algorithms see section 2.2. This part is usually crucial to the overall performance of the approach.

Standard vs. Meta-Search: Results for a query is typically gathered either by searching a local index (*standard* search) or by searching one or more other engines, combining the results, and using only the information returned by the other engines to do the clustering (*meta*-search).

Flat vs. Hierarchical Clustering: The clusters resulting from the algorithm can be either a flat list or a browsable hierarchy of two or more levels of clusters.

Single vs. Multiword Labels: Approaches differ in the way labels are constructed. This ranges from single word labels to lists of words or phrases.

On-Line vs. Off-Line clustering: On-line clustering approaches cluster the documents in a result set on-the-fly at query-time. Off-line approaches cluster all the documents at index-time, and only displays the pre-computed clusters at query-time.

3.2 Related Work

Clustering has been researched for decades, and since so many different strategies we can only present a selection of available clustering approaches.

Scatter/Gather: Scatter/Gather [12] was one of the first proposed clustering approaches intended for use on top of a search engine. It uses the non-hierarchical partitioning algorithms Buckshot and Fractionation [8] to cluster documents in linear time based on the cosine similarity between documents.

Grouper: Grouper [35] was one of the early approaches to web snippet clustering. It is a document clustering interface to the HuskySearch meta-search service, and it implements the Suffix Tree Clustering algorithm described in section 2.2.3. Heuristics based on word overlap and word coverage within each cluster are used to label the clusters with the best phrases identified by the STC algorithm. It is one of the best performing clustering engines, being able to cluster 500 clusters in only one second [35]. The system is available through the open source project Carrot2 [33].

Lingo: The Lingo system [17] uses Singular Value Decomposition (SVD) on the term-document matrix to find multiple word labels. It starts by identifying key phrases and represents them in the same vector space as the documents. Vectors are then transformed using SVD, and clusters are identified by using the notion of document similarity from the Vector Space Model, labelling the clusters with the terms closest to the center of the documents vectors in the cluster. This approach does not scale very well because SVD is a rather time-consuming process.

Clusty/Vivisimo: The Clusty/Visvisimo engine [22, 31] is one of the best performing commercial clustering engines available. It produces high quality hierachical clusters with multiple word or phrase labels. Clusty uses a meta-search approach drawing snippets from 10 other search engines. Vivisimo is an enterprise search platform that includes several services in addition to clustering. Little is known about the internals of the engine because it is commercial.

SnakeT: SnakeT [9] is a meta-search engine that draws results from 15 commodity search engines and builds hierarchical clusters based on snippets. It uses an on-line hierarchical clustering approach based on the notion of *frequent itemsets* from Data Mining. It extracts so-called *gapped-sentences* (related but not necessarily continuously appearing words) that form multiple-word labels. It also employs two knowledge bases (the Dmoz hiearchy and "anchor texts") to improve performance. It produces clusters of quality comparable to that of Vivisimo, but fails when data sets grow large. Experiments done in [9] show that the clustering 200 snippets takes over 4 seconds, and 400 snippets takes about 9 seconds, which is too slow for the average internet user.

Part III

Prototype

Chapter 4
Approach

This chapter describes the approach taken to document clustering, focusing on the choice of techniques. The next chapter describes the implementation of the different components in detail. The first section of this chapter presents the general idea and overview for the approach. The next section describes each of the components in greater detail and provides rationale for choices made at each level.

4.1 General Idea

The goal of the prototype is to be able to cluster web search results efficiently while producing high-quality labels. After reviewing the state-of-the-art and testing several of the clustering techniques, the Suffix Tree Clustering algorithm was chosen at the core of the approach. The algorithm proved to be much faster than the other approaches that were tested and it has several other attractive features (see section 4.4.5 for details). Since the clustering relies on snippets generated by the search engine, it is necessary to be able to control how snippets are generated. This means implementing a search engine from scratch, which gives the additional advantage of having complete control over the ranking.

With this in mind, a relatively standard search engine using classic IR methodology was developed. The clustering component was built on top of this, decoupled from the engine. The clusterer essentially only takes a set of snippets as input and returns clusters. Thus, it can easily be customized to fit on top of any engine.

4.2 Prototype Overview

The prototype operates in two separate phases. The first phase called 'Indexing' processes a document collection and builds an index to enable searching. The second phase called 'Retrieval' allows users to submit queries and then uses the index to retrieve relevant documents. The results are clustered and the user is presented with the clustered results. The subsequent sections describe these phases in greater detail.

4.3 Indexing Phase

Figure 4.1 shows the steps involved in the Indexing phase. The document collection is first pre-processed and then indexed. The index is stored on disk. The following sections detail each stage.

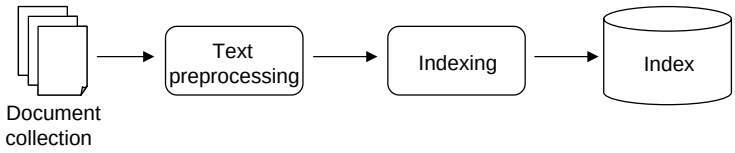

Figure 4.1: Indexing

4.3.1 Text Pre-processing

This stage prepares the documents for indexing by transforming them from a stream of characters to a list of index terms. At this stage a simple tokenization algorithm is first applied, and stopwords are then removed.

Stemming has been proposed as a means of reducing the index size and improve retrieval performance. Kraaij and Pohlmann [15] conclude that stemming does improve recall, but an evaluation performed by Hull [13] found little improvement in precision as compared to not using stemming. In this prototype the Porter stemming algorithm [18] is applied to the text, in order to reduce the index size and hopefully improve recall.

4.3.2 Indexing

Indexing is done by parsing the tokenized documents, keeping a list of each unique word (called the vocabulary) and a list of occurrences (called postings) to keep track of how many times a word occurs in each of the documents in the collection. This is done to create the classic *inverted index*. Before writing the index to disk, words that only occur in a single document are removed. In addition to the vocabulary and postings, certain features are extracted and calculated from each of the documents and stored in a separate file.

4.4 Retrieval Phase

Figure 4.2 shows the steps involved in the Retrieval phase. The search engine reads the necessary information from the index at the preparation stage. The user query is pre-processed and passed on to the search engine. The retrieval stage searches the index using the user query and returns a list of results. Snippets for the results are drawn from the document collection. The results are then clustered and presented to the user. The following sections detail each of these stages.

4.4.1 Preparation

The preparation simply consists of reading the vocabulary and document information from disk. There is no setup required for the clusterer.

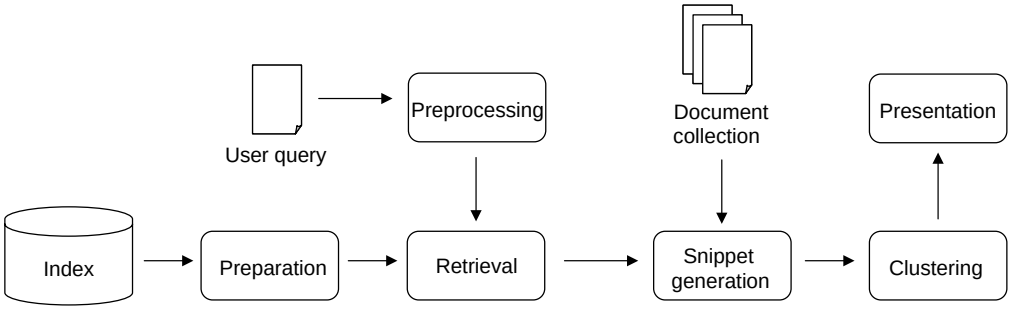

Figure 4.2: Retrieval

4.4.2 Query pre-processing

Queries are pre-processed in exactly the same manner as the documents, using tokenization, stopword removal and stemming.

4.4.3 Retrieval

The underlying search engine uses a classical approach. It uses the vocabulary and the inverted index built in the first phase. Term weights are calculated according to the well-known $td * idf$-scheme, and term weights for queries are calculated using the measure suggested by Salton and Buckley in [19]. Document similarity is calculated using cosine similarity.

4.4.4 Snippet generation

To speed up the clustering process, only a snippet of each document is considered. After ranking the result from the search engine, snippets from a number of the top resulting documents are generated. This is done simply by reading a portion of the original file from disk.

Using snippets instead of entire documents, not only reduced the workload for the clustering algorithm drastically, it also takes load of the usually very busy search engine core (that has to produce the snippets anyway). Relying only on snippets also allows clustering to be added on top a regular search engine to create a *meta-search engine* that uses the results and snippets returned by the search engine to do clustering.

4.4.5 Clustering

The results from the search engine are clustered using the Suffix Tree Clustering algorithm described in section 2.2.3. The algorithm takes as input the generated snippets, and returns a list of labelled clusters of documents.

The original paper describing the algorithm [16] lists several key requirements for web document clustering, including:

Relevance: The clusters should be relevant to the user query and their labels easily understood by the user.

Overlap: Because documents can have several topics, it is favourable to allow a document to appear in several clusters.

Snippet-tolerance: In order to be feasible in large-scale applications the algorithm should be able to produce high-quality clusters based only on the snippets returned by the search engine.

Speed: The clusters should be generated in a matter of milliseconds to avoid user annoyance and scalability issues.

The STC algorithm has all these qualities, but the main reason for choosing it was its speed, simplicity and ability to produce good cluster labels. Several factors influence the performance of STC, and the following sections describes the choices made at each step:

Stemming Snippets

[34] suggests stemming all words in the snippets with a light stemming algorithm. To present readable labels to the user they save a pointer for each phrase into the snippet from which it originated. In this way one original form of the phrase can be retrieved. Note that several original phrases from different snippets may belong to the same base cluster, but which of these original forms is chosen is not detailed in the article.

The prototype only stems plurals to singular, using a very light stemming algorithm. The effects of stemming the snippets is further assessed in the evaluation (see section 7.2.2). Original snippets are stored to allow original phrases to be reconstructed after clustering.

Removing Stopwords in Snippets

The authors of STC suggest dealing with stopwords in phrases by allowing them as long as they are not the first or the last word in the phrase, for instance allowing 'product of France' but not 'product of' [35].

Testing the prototype without removing stopwords, it seems that phrases containing stopwords are rarely selected as labels. Therefore, the prototype simply skips stopwords and inserts phrase boundaries instead. Testing indicates that this has very little impact on the resulting clusters and it's therefore preferred for it's simplicity.

Labelling Clusters

The clustering algorithm outputs a set of labelled base clusters for each cluster, and the authors suggest using these base cluster labels as labels for the final cluster. In their Grouper system, clusters are labelled with all the labels from the base clusters [35].

The STC algorithm assigns a score to each base cluster, but never utilizes it. The approach taken in the prototype is to treat the base cluster labels as *candidate labels* for the final cluster, and use the scoring to select the highest ranked candidate as the final label. It is assumed that having one or two phrases as a label instead of labelling each cluster with *all* candidates enhances usability.

The original paper describing STC suggests scoring base clusters using the formula $s(B) = |B| * f(|P|)$ where $|B|$ is the number of documents in base cluster B, and $|P|$ is the number of words in P that have a none-zero score. Zero-scoring words are defined to be stopwords, words that appear in less than three documents or words that appear in more than 40% of the document collection. f is a function that penalizes single word phrases, is linear for two to six words long phrases and constant for longer phrases.

The scoring scheme used by the prototype closely resembles the suggested formula, because initial testing indicated that it worked very well. Whether the algorithm succeeds in selecting the "correct" candidate is addressed in the evaluation chapter section 6.3. The prototype doesn't include stopwords in the zero-scoring words since all stopwords are removed during pre-processing. The function f gives a score of 0.5 for single word phrases, 2 to 6 for two to six words phrases respectively, and 7 for longer phrases. If the two top ranked base clusters have the same score, the one with the largest document set is used as the label. This is because phrases occurring in many documents are assumed to represent more important categories.

Ranking Clusters

As previously mentioned, the original STC paper does not detail how the final clusters are ranked. One of the best performing systems available, Clusty.com, seem to almost always presents cluster lists sorted by the number of documents they contain. Minor deviations from this approach can be observed, indicating that they use some additional heuristic. This suggests that it might be a small trade-off between quality and performance, given that additional processing of the clusters can produces a slightly altered (and hopefully improved) ordering of clusters. In the prototype this extra processing is assumed to cost more time than it gains in quality. It is assumed that large clusters are more interesting, and clusters are thus simply ranked by the number of documents they contain.

4.4.6 Presentation

The prototype includes a web front-end to the search engine merely for testing purposes. It allows the user to submit a query and returns a list of matching documents and browsable clusters. The documents are presented with the document title, the generated snippet and a link to the corresponding web page.

4.5 Gathering Test Data

For evaluation purposes, a collection of web documents was needed. A crawler was developed to generate the document collection. It takes as input a set of queries and uses the Google Search API [24] to search Google. For each query, the URLs of the 1000 top hits are stored and the corresponding web pages are downloaded. The documents vary in length but are on average about 5 kilobytes after stripping the html codes. For English text, this is about 800-1000 words.

To reduce the space requirements and remove noise from the downloaded documents a good HTML-stripper was needed. Several open source variants were tested, but none of these managed to deal with the noise and errors present in HTML code on the web. Therefore a simple yet efficient stripper was developed from scratch.

CHAPTER 5
IMPLEMENTATION

The prototype was developed in using Sun's Java 1.4.2 SDK. It consists of a set of a crawler for gathering documents, the search engine with clustering capabilities and a web front-end to present the results to the user.

The implemented prototype operates in the two phases described in the previous chapter. The subsequent sections describe the implementation of these phases. These sections mirror those of the previous chapter.

5.1 Indexing Phase

This section provides implementation details for the Indexing phase. Figure 4.1 showing the steps of the Indexing phase is repeated here for convenience. The following sections describe the implementation of each stage.

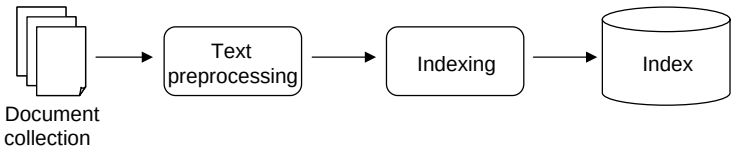

Figure 5.1: Indexing

5.1.1 Text Pre-processing

Several pre-processing steps transform each document from a stream of characters to a set of index terms.

The document is first tokenized, by means of a simple tokenization algorithm. Web documents tend to contain a lot of noise and special characters so the index is easily filled with terms not useful for retrieval. To reduce noise tokenization is achieved simply by selecting all sequences of standard English letters (A-Z or a-z) of more than one character. This approach has is obvious limitations (such as not being able to index words with special characters or text in foreign languages properly), but is chosen because of its simplicity and the fact that all documents in the collection are in English.

The pre-processor is provided with a file containing stopwords to be removed from the index. All tokens that appear in a list of stopwords are removed, and the cleaned set of tokens is passed on to the next stage.

For all tokens a filepointer to its position in the original document is also stored to aid operations such as snippet generation.

The prototype uses the Porter stemming algorithm, and a java version of the stemmer is available at Martin Porter's homepage [29].

5.1.2 Indexing

The Indexer processes each word of each document, and keeps track of the number of times it appears in each document. This vocabulary is first sorted, and then a list of document IDs and word frequencies is written to a *postings* file for each of the word. In a separate file (called the *vocabulary* file) the word is written along with the number of documents it appears in and a filepointer into the postings file. Afterwards, vectors containing the vector lengths of all document vectors and the frequency of the most frequent word in the document are built. These are needed for the similarity computation at query time. All information about the documents are written to a *document info* file. This includes an ID, the file name on disk and the title and URL for the corresponding web page.

The index is capable of indexing at least a couple of hundred thousand document just using main memory available at most desktop computers today. If a larger index needs to be constructed, one could easily build several partial indices the size of the main memory, store them on disk, and then merge them afterwards [4].

5.2 Retrieval Phase

This section provides implementation details for the Retrieval phase. Figure 4.2 showing the steps of the Retrieval phase is repeated here for convenience. The following sections describe the implementation of each stage.

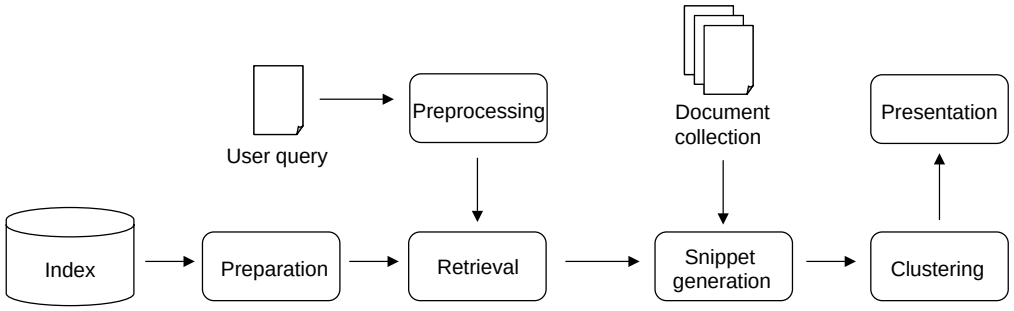

Figure 5.2: Retrieval

5.2.1 Preparation

Preparing the search engine to accept queries consists simply of reading the vocabulary and the document information stored on disk by the Indexer. The vocabulary is a set of triplets

called VocabularyEntries containing the actual word, accompanied by the number of documents containing the word and a filepointer into the postingsfile. Keeping the vocabulary sorted allows us to binary search it without any excess memory usage, yielding a runtime complexity of only $O(lg\ n)$. The documents are stored by a DocumentManager, that allows easy access to auxiliary file information such as document title, url and information needed by the similarity calculations (document vector length and the frequency of the most frequent word in the document).

Since the vocabulary and document information files are small compared to the size of the collection, they usually fit in main memory even for large collections. Testing done with a 250MB collection of 50000 documents resulted in vocabulary and document information files that were just over 10MB combined. The size of the document information file is obviously linear to the collection size. The vocabulary on the other hand grows sublinearly according to *Heaps' Law* [11]. The law states that vocabulary size $V = O(n^{\beta})$, and experiments indicate that β is between 0.4 and 0.6 for the TREC-2 collection [2]. This means that the memory requirements grow sublinearly to the collection size, thus suggesting that document information and vocabularies can be held in main memory even for very large collections.

5.2.2 Query pre-processing

Given a query, the engine pre-processes it using the same techniques applied to the documents in the pre-processing stage of the Indexing phase. That includes tokenization, stopword removal and stemming, see section 5.1.1 for further details.

5.2.3 Retrieval

For each of the query tokens the term weight is calculated according to the formula in 2.1, and the corresponding DictionaryEntry is retrieved. Using a sparse vector representation for storage, term weights are calculated for the query terms for each of the documents in the index according to the $tf * idf$-measure. The cosine similarity is then calculated between the query and all documents in the index. Since the DocumentManager stores vector lengths and highest frequencies for each document the similarity calculation requires only one disk access per token in the query, which is acceptable because user queries tend to be short.

5.2.4 Snippet generation

The DocumentManager is also responsible for generating snippets. For each word in the vocabulary, a filepointer is stored to its first occurrence in each document. Snippet generation is simply done by reading the first 150 characters from the first query term occurring in the document onwards. The file pointers use a block addressing scheme roughly mirroring sentence structure so that snippets are usually drawn from the beginning of a sentence.

5.2.5 Clustering

A suffix tree the uses words instead of the usual characters as nodes was implemented. The tree is built from a list of phrases from each document. In order to reconstruct these phrases to display as labels the original phrases are stored and the pre-processed phrases each map to

one original phrase. Each pre-processed phrase is a String array, and the label for each node in the tree is represented by one of these phrases, an offset and a length. This allows to use only use pointers to the original strings and two integers for each node in the tree, both saving memory and eliminating the need for copying and changing strings.

Although linear time suffix tree construction algorithms exist, these are often cumbersome to implement. As the suffix tree construction was not found to be a bottleneck in the implementation, a naive algorithm resembling how one would construct a suffix tree by hand was used.

5.2.6 Presentation

The web front-end was developed using the Spring open source framework for Java [30]. Spring provides a comprehensive Model-View-Controller framework for Java Server Pages. The web application was deployed on the open source application server JBoss.

Figure 5.3 shows the user interface presenting the results of a query and the clusters.

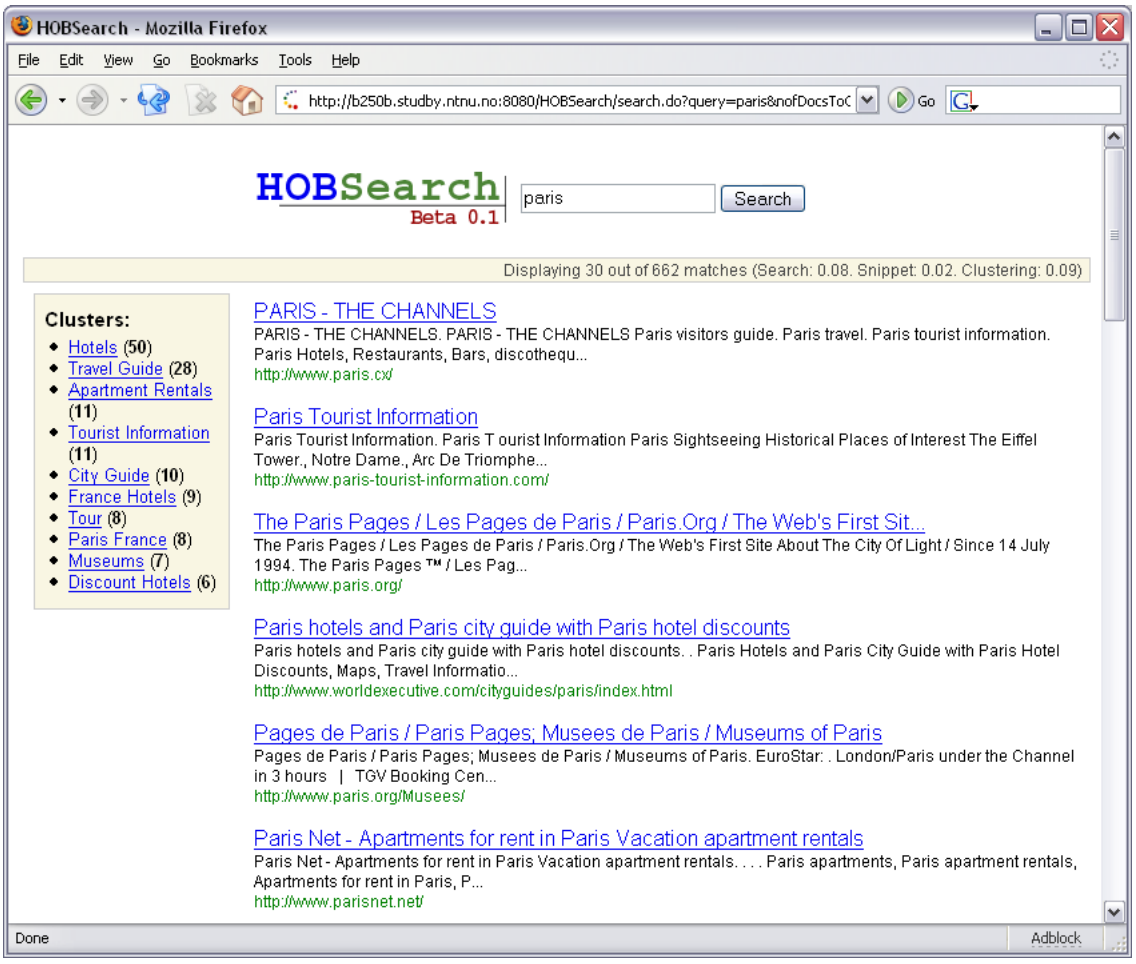

Figure 5.3: Web Interface

5.3 Class Diagrams

This section details the actual implementation of the prototype. Section 5.3.1 describes the five top-level packages, and sections 5.3.2 and 5.3.3 describe the internals of package `searching` and `clustering` respectively. Since the other top-level packages (`web`, `tools` and `crawling` only consist of independent classes their internals are only briefly discussed in the next section.

5.3.1 Packages

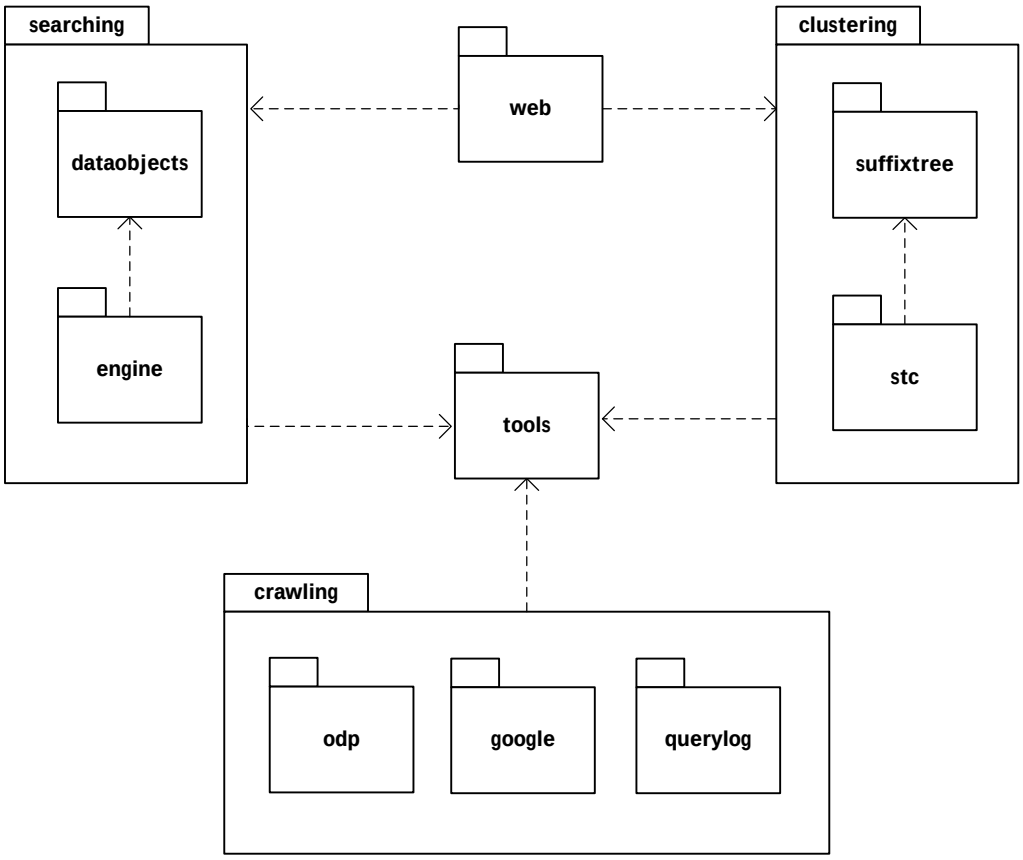

Figure 5.4: Packages

Figure 5.4 shows all packages in the prototype and their dependencies.

Package `searching` is responsible for the search engine functionality such as building and index and allowing search. It consists of the `engine` and a set of `dataobjects` used for storing data by the engine.

Package `clustering` implements the document clustering. Package `stc` implements the Suffix Tree Clustering (STC) algorithm and depends on the `suffixtree` package for constructing the suffix tree needed for storage.

Package `crawling` contains several independent utility classes used to build the document collection and to extract information from the ODP hierarchy and the query log. Package

`google` queries Google through the Google API, and allows the resulting urls to be downloaded to disk. Package `odp` traverses the ODP hierarchy to extract labels. Package `querylog` analyses the original query log, extracts only the actual queries and performs simple text processing to "clean" the data. The `crawling` also depends on the `tools` package.

Package `tools` provides shared tools used by the other packages. These tools include an `HtmlStripper`, a `PorterStemmer`, a `StopWordRemover`and other convenience classes for performing simple operations.

Package `web` provides the web front-end. It consists of a single servlet called `MainController` that acts as the "Controller" and builds the "Model" in the MVC-paradigm. The views are provided by a set of Java Server Pages not shown in the diagram.

5.3.2 Package Searching

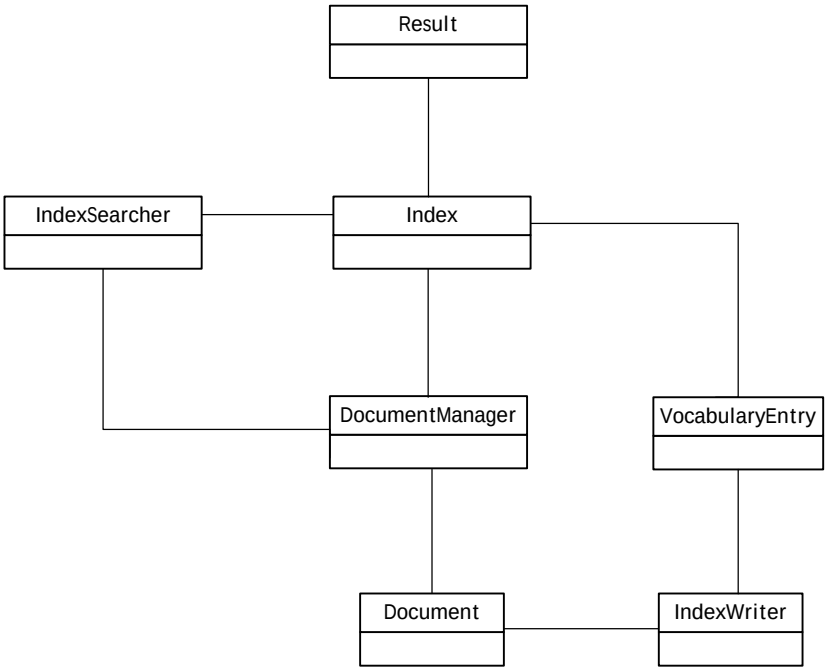

Figure 5.5: Package Searching

Figure 5.5 shows the internals of the `searching` packages. The `IndexWriter` is responsible for constructing the index and it uses `Document` and `VocabularyEntry` to hold document and vocabulary information while indexing. The `DocumentManager` controls the document collection and provides the `Index` with information about each document. The `Index` is the class responsible for look-ups in the data-files containing the index on disk. It stores the vocabulary as `VocabularyEntries`, accepts queries and produces lists of `Results`. The `Result` class is a passive data object containing information about a specific query result such as document title, similarity with the query etc. `IndexSearcher` is just a wrapper class providing a convenient way of querying the `Index` , gathering the `Results` and retrieving document information from the `DocumentManager`.

5.3.3 Package Clustering

Figure 5.6 shows the internals of the `clustering` packages. The `SuffixTreeClusterer` is responsible for the actual clustering, storing temporary data in `BaseCluster`s and final clusters in `Cluster`s as approperiate. It depends on the `SuffixTree` for constructing a suffix tree from the snippets. The tree consists of a hierarchy of `Node` objects, that can be either `InternalNode`s or `LeafNode`s.

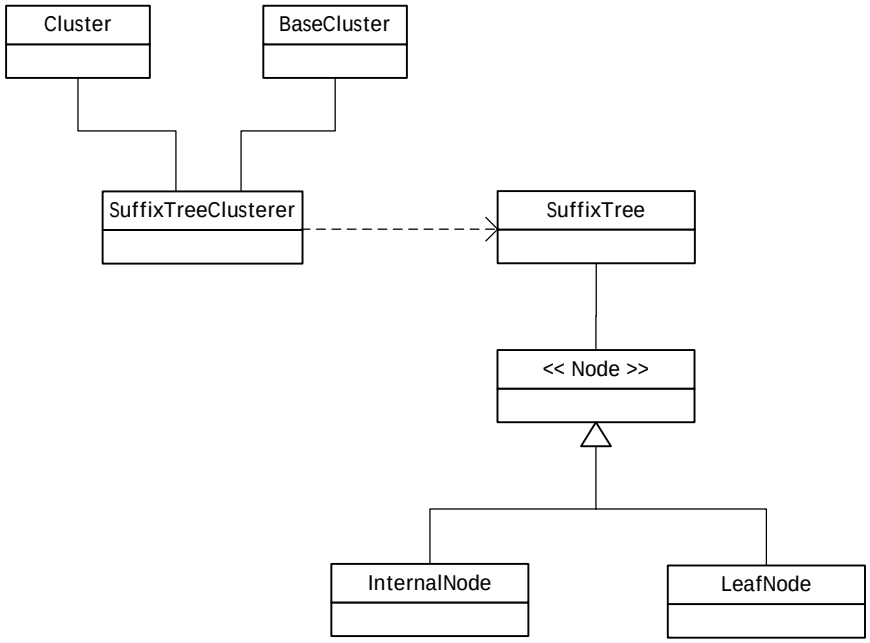

Figure 5.6: Package Clustering

Part IV

Evaluation

CHAPTER 6
EVALUATION

This chapter describes the formal evaluation of the prototype. Two tests were set up, and the following sections provide details about the tests, methodology and results.

Several aspects of the prototype have been assessed only through experimentation with the prototype code. No specific test setup or methodology was used, and the results are therefore only included as a discussion in the next chapter.

Since we seek to achieve performance comparable to that of the state-or-the-art the best thing would of course be to evaluate directly against some of the best available systems. However, since these systems are commercial we could obviously not get hold of the same document collections and other resources, making a direct comparison very difficult. The STC algorithm lies at the core of our approach, and has been evaluated against many other well known clustering algorithms in [34].

It is difficult to evaluate cluster labels directly, because it ideally involves investigating the contents of potentially hundreds of documents in each cluster. Since we have limited evaluation resources the quality of the cluster labels is assessed only through inspection of the candidate labels and example document titles resulting from the STC algorithm.

This chapter describes the document collection and the test setup used for all evaluations, and the last sections detail the formal evaluation.

6.1 Document Collection

Using the crawler described in section 4.5, 1000 documents were retrieved from Google for each of the queries used in the evaluation. The queries are 'paris', 'jaguar', 'apple', 'hollywood' and 'red hot chili peppers'. All documents containing less than twice as many characters as the document title were discarded. This is because many web sites simply consist of a flash animation or a splash screen that is not interesting for retrieval purposes. Some documents were not retrieved because of technical issues such as web servers not responding et cetera. A total of about 3900 documents constitute the final document collection.

The language in the documents is obviously dependent on the queries used to retrieve them, but for the queries used in this document collection the language is assumed to be rather unformal and without many technical details.

6.2 Test Setup

The tests were run using the document collection described above, and the prototype was set up according to the approach described in chapter 4. This involves removing stopwords and stemming the words in the index, and removing all words that appear in only one document. The snippets are generated by reading 150 characters from the sentence in which the

first query word appears in the original file onwards. Stopwords are removed from the snippets and trailing s'es in plural forms are removed. Candidates are scored using the standard method suggested by [34] and base clusters are joined when they have a mutual document overlap of more than 40%. Each cluster is labelled with its highest scoring base cluster.

6.3 Label Quality

In essence, the STC algorithm outputs a number of clusters that contain base clusters, whose labels can be used as candidate labels for the cluster. A crucial step in the algorithm is thus the selection of one or more of the base cluster labels to serve as the cluster label, since this greatly impacts the user experience. Given a set of candidate labels, this test evaluates whether our approach correctly selects the best candidate to serve as a label for each cluster.

6.3.1 Methodology

This test was performed by four computer science students (three master degree candidates and one PhD candidate). Five different queries where run and the first 200 resulting documents were clustered. The queries were 'paris', 'jaguar', 'apple', 'hollywood' and 'red hot chili peppers'. The test subjects were presented with all candidate phrases (in random order) of the top ten ranked clusters for each query. They did not know which of the candidates that were selected as the cluster labels by the system.

Since some clusters are larger than 50 documents, having the test subjects sift through all documents for all clusters would be too laboursome. Instead, they were provided with five *example document titles* for each cluster and the *coverage* of each candidate phrase. The coverage is defined as the percentage of documents in the cluster in which the candidate phrase occurs.

For each cluster, the test subjects were instructed to select the most approperiate candidate to serve as a cluster label. In addition, they gave a score for *cluster coherence* and *label quality*. Cluster coherence is whether the candidate phrases and example document titles seem to describe a common topic. Label quality was defined as whether they think the candidate they chose as cluster label is a suitable cluster label. The scores were given using a three point scale (poor, medium and good). The evaluation forms and instructions sent to the test subjects can be found in appendix A. This appendix also lists all phrases generated for all five queries, and shows which ones were chosen as labels by the prototype.

6.3.2 Expected Results

The document clustering quality (coherence) is assumed to be good, since the mechanism for grouping documents essentially is the same as in the orginial paper on STC. Initial testing of the prototype indicated that the most approperiate candidate label was often chosen by the prototype, so the test subjects are assumed to have a relatively high percentage of correctly chosen candidate labels. The label quality might vary significantly among the test subjects since it's the most subjective part of the evaluation.

6.3.3 Results

Figure 6.1 shows the average score given for each query (see appendix B for details about the results for each test subject). Note that precision ranges from 0 to 1 while cluster coherence and label quality range from 0 to 2. Precision is defined to be 1 if the subject selected the same candidate as the prototype and 0 if another candidate was selected. Some of the test subjects often selected a candidate equal to the one selected by the prototype, but prefixed by the query (for instance selecting *Paris Hotels* instead of *Hotels* for the query 'paris'). In these cases a score of 0.5 was assigned. Whether to allow the cluster labels to be prefixed by query or not is further addressed in section 7.2.3.

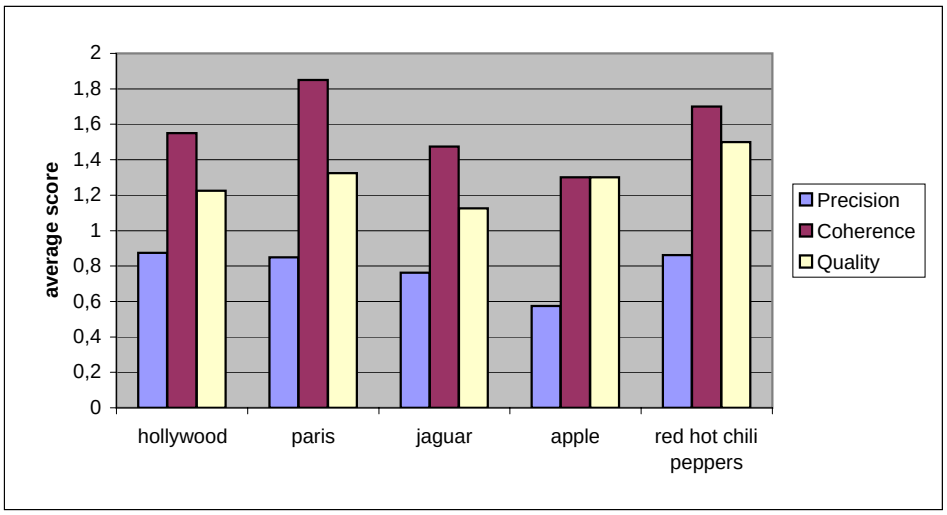

Figure 6.1: Average score for all queries

Since only four test subjects were used, variability in the results is expected. It is interesting to see which of the dimensions (precision, coherence, quality) that vary the most. Figure 6.2 shows the total number of points assigned by each of the subjects for the three dimensions. Again, refer to appendix B for all result details.

6.4 Clustering Performance

In order to be feasable in a large-scale setting, the prototype should be able to cluster hundreds of document snippets in less than a second. This test addresses the performance of the prototype when clustering increasing number of snippets.

6.4.1 Methodology

Using the query 'paris' the execution time of the clustering operation for the prototype was measured for various collection sizes. Tests were run with 100 to 1000 snippets, and each reported time is averaged over 100 tests for each collections size.

Time is measured from the search and snippet generation has completed, until the final clusters are created and labelled. This is done to emphasize the overhead of the clustering,

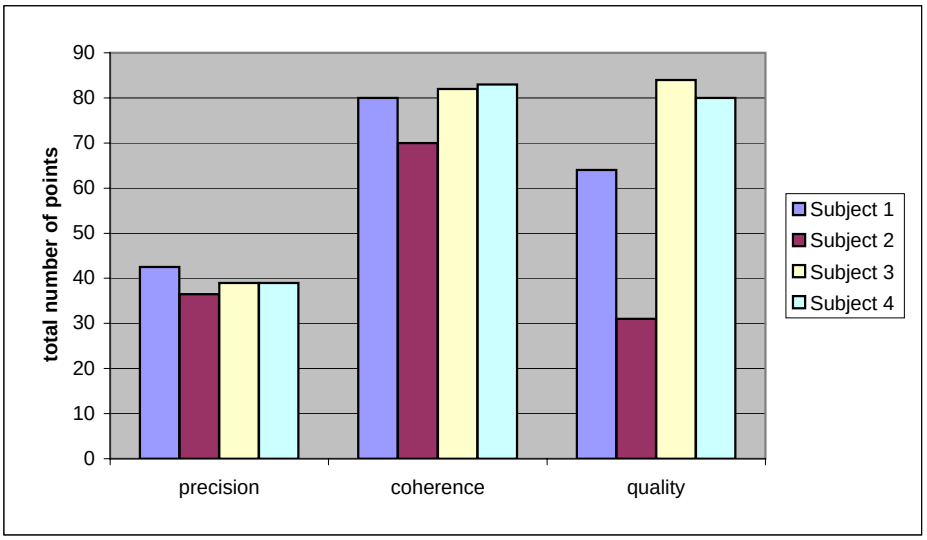

Figure 6.2: Total score for each test subject

and to give an idea of how fast it would be when used in a meta-search approach (i.e. drawing snippets from an external search engine).

Tests are run on a Athlon XP 1700+ (1,47GHz) with 512MB RAM. The execution times are compared to those reported by [34] using the orinal STC algorithm implementation. It's important to note that these tests were run on a Pentium 200 (200MHz), thus making the results hard to compare.

6.4.2 Expected Results

Even though the authors claim the execution time of STC is linear to the collection size, the prototype is not expected to perform linearly. This is in part because the suffix tree construction, which is an important part of the overall time spent on clustering, is not done in linear time. In addition, several parts of the prototype have not been tuned optimally, further degrading the performance.

The clustering is expected to be feasable for up to about 500 document snippets, meaning that it should not take more than about a second to perform.

6.4.3 Results

Figure 6.3 shows the execution time (in milliseconds) of the prototype clustering algorithm and the original STC implementation as a function of collection size.

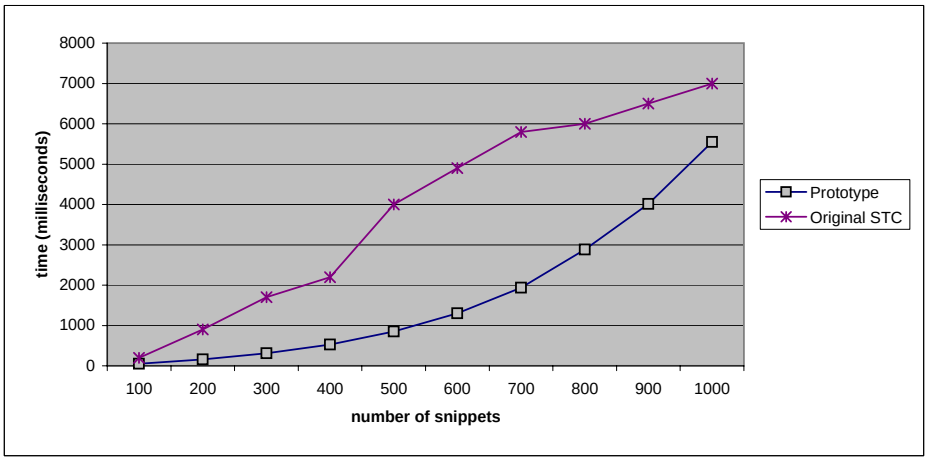

Figure 6.3: Clustering Performance

CHAPTER 7
DISCUSSION

This chapter discusses the results of the evaluation, interesting observations made while experimenting with the prototype and possible improvements of the prototype.

7.1 Findings

Section 6.3.3 and 6.4.3 present the results from the formal evaluation of the prototype. This section investigates these results and presents the findings.

7.1.1 Label Quality

The average number of times where the prototype selected the same label as the test subjects is close to 80%, which is quite impressive. It indicates that the prototype in most cases succeeds in selecting the same label as human users. However, this does not mean that it necessarily selects the *correct* label, but merely the same as a small sample of test users.

The average score for the coherence of each cluster is 1.6 out of 2 points, which means that they on average are closer to "good" than "medium". This indicates that the clustering algorithm is good at grouping documents, and it confirms the original assumption that documents that share phrases often address the same topic.

Results were less promising when it comes to label quality, averaging at only 1.3 out of 2 points. Looking at the results from each test subject (appendix B), we discover that three subjects have an average of about 1.5 while the last subject has an average of only 0.6. Although the points for quality were expected to vary between subjects, this is suprisingly much. It indicates that label clustering is generally very subjective, but also that more test subjects should have been used to reduce the effects of potential 'outliers'.

It is interesting to look at what causes low precision for some of the clusters. Looking at the average precision from appendix B we notice two cases in which the score is particularly low: when clusters have many overlapping candidates and when they have little coherence. Both cluster 6 of the query 'paris' and cluster 3 for 'red hot chili peppers' score below 0.5 points, and have many candidates that are highly overlapping. Having many similar candidates naturally makes it difficult to choose the correct label because users might find several labels equally good. The low precision is therefore to be expected. As for the other set of clusters with low precision see for example cluster 6 from 'jaguar' and clusters 1 and 2 from 'apple'. These clusters all have many candidates and in addition to having little or no coherence.

Looking at the average scores and the number of candidates for each cluster, it is obvious that the method of evaluation has it's limitations. If all clusters had only one candidate they would all have 100% precision and nearly 100% coherence, although label quality might have declined.

As described in the result section of the previous chapter, two of the test subjects repeatedly market the same candidate as the prototype but prefixed with the query (for instance marking *Red Hot Chili Peppers Concert Tickets* rather than *Concert Tickets* for the query *red hot chili peppers*). We argue that in most cases labels should not be prefixed by the query. It is assumed that if the test subjects where shown a list of only the final cluster labels along with the query (much like in the web interface) they would recognize the redundancy in prefixing each cluster with the query. Had the instructions (see appendix A) given to the test subjects adressed this issue, which they unfortunately do not, the results might have been different.

7.1.2 Clustering Performance

The performance test was done simply to show that clustering on-the-fly is feasable for several hundred documents. Figure 6.3 shows that up to 500 documents can be clustered in under a second. Testing has given good results when clustering only 200 documents, and that took only 160 milliseconds on average. In addition, the clustering can be built outside the search engine core (a meta-search approach) without impacting the performance of the search engine.

As previously mentioned, the original STC implementation was tested on a Pentium 200MHz and the prototype on a AMD Athlon 1700+. That makes the two curves hard to compare directly. It is interesting to see that while the original implementation seems more or less linear to the collection size, the curve for the prototype forms a parabola. It is difficult to say exactly what separates the two, but the original implementation is obviously much better tested and tuned. In addition, we know that the suffix tree construction algorithm used in the prototype is not linear.

7.2 Prototype Experimentation Findings

The prototype has many features and parameters that can be tuned and altered. This section presents the informal testing done through experimentation with the prototype.

7.2.1 Base Cluster Similarity

The original paper on STC joins base clusters that have more than 50% mutually overlapping documents (see the formula in section 2.5). Through manual inspection of the base clusters contained in each of the final clusters for several queries, it seems that many clusters are so similar that they should have been joined. A solution might be to lower the threshold to increase the number of base clusters joined, thus creating fewer and larger clusters. Table 7.1 shows the clusters with all base clusters for the query 'paris' when clustering 200 documents and using a 50% threshold.

Experimenting with the prototype indicates that 40% might be a better threshold. Contrary to the findings of the authors, it seems that this factor impacts clustering performance a great deal. A mere 10% change from 50% to 40% resulted in a reduction from 136 to 96 clusters (30% fewer) when clustering results from the query 'paris' using 200 snippets. Table 7.2 shows the results when using the 40% threshold. It is difficult to quantitatively measure any improved quality resulting from this, but the average number of base clusters in the top 20 ranked clusters increased from 1.6 to 2.6 when lowering the threshold. Of the final clusters, 'Travel' and 'Guide' are joined into the cluster 'Travel Guide', and 'Hotels' and 'Hotels Paris'

Cluster	Phrases
1	{France}
2	{Hotels, Paris Hotels}
3	{Travel}
4	{Guide}
5	{Hotels Paris}
6	{Pages}
7	{Tour}
8	{City}
9	{Paris Apartment Rentals, Apartment Rentals, Apartments, Rentals, Paris Apartments}
10	{Travel Guide, Paris Travel Guide, Paris Travel, Restaurants}

Table 7.1: Base Clusters with 50% threshold

are joined into the cluster 'Hotels'. Both of these changes seem attractive. In general, having a 40% threshold results in more clusters with only one base cluster.

Cluster	Phrases
1	{France}
2	{Hotels, Paris Hotels, Hotels Paris}
3	{Travel Guide, Paris Travel Guide, Travel, Guide, Paris Travel, Restaurants}
4	{Pages}
5	{Tour}
6	{Paris Apartment Rentals, Apartment Rentals, Apartments, Rentals, Rent}
7	{City}
8	{Information}
9	{Paris France}
10	{Hotel Reservation, Paris Hotel Reservation, Reservation}

Table 7.2: Base Clusters with 40% threshold

7.2.2 Effects of Stemming Snippets

The authors of the STC algorithm suggest stemming the snippets in the pre-processing stage. Perhaps the most popular stemmer available is the Porter stemming algorithm [18]. Since we are mostly interested in nouns, Porter's algorithm is a bit cumbersome. Experimenting with the prototype revealed that stemming takes about 200 milliseconds when clustering 100 snippets, and it grows approximately linearly with larger numbers of snippets.

It is natural to assume that the additional cost of stemming would to some degree be made up for by the reduced size of the suffix tree and thus faster computation of clusters. A test done with 200 snippets from the query 'paris' showed that 2058 tree nodes where created without stemming and as much as 2025 nodes when using stemming. Surprisingly, it seems that stemming has little effect on the size of the tree.

This supports the idea of using only a very light stemming algorithm. In the first attempt, plural endings for nouns (such as *-s* and *-ies*) were stripped. Testing revealed that as much as 99% of the endings stripped were trailing s'es. To further increase the speed of the stemming

algorithm, it was therefore reduced to simply stripping the s if it's at the end of a word and the second last character is not an s.

In general, stemming does not seem to influence the clustering significantly. Several tests indicate only minor changes in the top 10 cluster labels and about 10% reduction in the total number of clusters.

7.2.3 Label Overlap

Because the all suffixes of each phrase is generated, it is common that clusters contain several very similar phrases (e.g. *jaguar car reviews*, *car reviews* and *reviews* for the query 'jaguar'). Often the phrases start with the query itself, but it seems that the final labels seldom contain the query. This is because the query terms get zero score since they appear in more than 40% of the collection. This is a good thing, since we are usually not interested in having all clusters prefixed by the query.

However, in some cases we do want the query to appear in the label. For instance when *Apple Macintosh* appears while searching for *apple*. A solution could be to use *Named Entity Recognition* to detect trademarks and people etc. One could then say that the word *apple* in a phrase does not recieve a zero score if the entire phrase is a *named entity* (e.g. when the next word is *macintosh*). That way, the phrase *apple macintosh* would recieve a greater total score than *macintosh* and thus be selected as the label.

7.2.4 Snippet Generation

Snippet generation requires reading from disk a portion of each of the documents that are to be clustered. This turned out to be a major part of the overall time spent for each request when testing on a single desktop computer. Time required to read a couple of hundred file portions varied from 0,2 seconds to 5 seconds, depending on several factors: whether the disk was currently busy with another operation, and the size of the folders containing the documents. The first factor is very hard to control when using a single desktop computer, but should be less crucial when using dedicated file servers. The second factor was found surprisingly important when testing on the Windows NTFS file system. It seems that directory look-ups get really slow when they contain several thousand files. The solution was to distribute the files in a large number of directories, which drastically improved the performance.

7.2.5 Parts-of-Speech in Cluster Labels

For descriptive phrases, we are generally most interested in nouns and adjectives. To achieve this, one might use a part-of-speech tagger and then select only spesific parts of speech to serve as label candidates. Unfortunately, part-of-speech tagging is generally very expensive. It is therefore interesting to observe that despite not using any part-of-speech detection, the resulting cluster labels are almost always nouns, sometimes adjectives, but rarely verbs or any other parts of speech. It seems that analyzing parts of speech is a waste of time when using the STC algorithm.

7.3 Improvements

Since limited time was available for developing the prototype, several features have been implemented in a simple and straight-foreward manner. The resulting prototype works, but is far from optimal. This section describes some potential improvements of the prototype.

7.3.1 Search Engine Ranking

In order to generate high-quality clusters for a given query, high-quality snippets are needed, which in turn depends on relevant and high-quality documents in the result set produced by the search engine. Ranking in the search engine is done simply by using the $tf * idf$ measure and computing the cosine similarity between the query and all documents. This can be improved through for instance employing the HTML-code available for each document to extract important features such as headlines. More sophisticated schemes that utilize link information like Google's PageRank [5] could also be implemented.

7.3.2 Snippet Generation

Since the clustering algorithm relies solely on the snippets, their quality is very important. Good quality snippet should contain as much information about the query and as little noise as possible. In the prototype, snippets are simply drawn from the first appearing query word onwards. This ensures the occurrence of the first query word, but not necessarily the best snippet for that query word and certainly not the best snippet if the query contains multiple words. A slightly more advanced scheme would for instance be to use the paragraph containing the maximum number of query words, or multiple small passages containing all query words.

7.3.3 Suffix Tree Tuning

Several techniques can be used to improve the performance of the suffix tree used for clustering. As previously mentioned, the prototype uses a naive construction algorithm, and the tree can be built in time linear to the collection size [20].

Since all phrases that occur in less than three documents are discarded, they could be removed before building the tree. Reducing the size of the tree makes building and searching the tree faster.

The data representation used for each node is not optimal. An obvious improvement would be to encode each stemmed word as an integer instead of a string, because matching integers is much faster than matching strings. [35] also suggest encoding documents belonging to each node with bit vectors to reduce time spent on calculating overlap between clusters.

CHAPTER 8
EMPLOYING ADDITIONAL KNOWLEDGE BASES

Although indicating that the acutal clustering works well, the evaluation of the prototype shows that labelling clusters with good quality phrases is very difficult. This chapter focuses beyond the actual clustering algorithm used, and discusses what kind of additional data sources could be useful in extracting high-quality labels for clusters. The last section discusses how these techniques apply to the STC algorithm used in the prototype.

8.1 Clustering using a query log

A problem with traditional document vector based clustering algorithms is how to extract meaningful labels. Assuming a very large log of queries submitted by users, this could actually serve as the user *vocabulary*. The assumption is that users describe their information needs using their perception of a domain rather than the contents of the documents describing the domain.

An idea is to translate the document vectors from the usual index term-space into a query log-space consisting only of those index terms contained in the query log. Then clustering could be performed as usual, but it would cluster based on the words used by actual users. This could hopefully yield more natural groupings of documents, reflecting how users percieve the domain. Also for the actual labelling, one could assume that terms appearing in a query log are more likely to make a good labels than just any word in a document.

8.2 Using the ODP hiearachy

The ODP hierarchy inhibits at least two valuable sources of information for clustering: the category labels are likely to be good labels for clusters since they often represent concepts, and the hierarchy structure can be useful for labelling hierarchical clusters.

The *SnakeT* system (described in chapter 3) utilized the ODP hierarchy for cluster labelling. It calculates a sort of $tf * idf$ score that reflects how significant a word is and how specific it is (measured by its depth in the hierarchy). They then use these scores to boost the words accordingly, thus increasing the likelyhood of selecting these words as labels for the clusters.

8.3 Improving the STC algorithm

Several test were carried out using both ODP data and a query log with the prototype. None of them showed any immediate improvement, but we argue that this is due to the nature of the STC algorithm rather than the knowledge bases. The main problem is that unless the fundamental clustering technique of grouping frequent phrases is altered, STC generates rather

few candidate labels for each cluster (typically less than five). In addition, these labels are very frequently subsets of each other since all suffixes of all phrases are generated.

Given a cluster with only one or two candidate labels, it is unlikely that ODP data or a query log can improve the labelling. Experimentation with the prototype indicated that to get a noticable change in the labelling as a result of for instance the ODP data, words appearing in the hierarchy must recieve a significant boost in their score. This did however seem to degrade the ranking of the labels rather than improve it.

Named Entity Recognition might on the other hand serve a special purpose in combination with the STC algorithm. The evaluation revealed that STC in most cases does not generate cluster labels that are prefixed by the actual user query, but in some cases this is exactly what we want. Given for instance the query *johnny* and a cluster with the candidates *Johnny Cash* and *Cash*, the former is more likely to be relevant even though it is prefixed by the query. This represents a special case where the phrase 'Johnny Cash' represents a *named entity*. Under normal circumstances, the first phrase would not have recieved a higher score than the second because johnny would appear in more than 40% of the snippets and thus be considered a 'stopword'. If such named entitites are recognized, they can be boosted to increase the likelyhood of being selected as cluster labels. In general, it seems that phrases representing named entities are more likely to make good cluster labels.

Chapter 9
Conclusion

Clustering in a data mining setting has been researched for decades. Lately, document clustering used to cluster web search engine results have recieved much attention. Large companies such as Google and Microsoft have shown their interest and we have seen the emergence of commercial clustering engines such as Vivisimo.

This thesis has shown how a search engine with clustering capabilities can be developed. The approach described has been implemented as a working prototype that allows searching and browsing clusters through a web interface. The prototype has been evaluated in a user survey and through informal testing.

The evaluation indicated that the resulting clusters are coherent and that clustering several hundred documents is feasable. Unfortunately, the quality of the cluster labels was found to be only "medium" by the test subjects. It is however believed that additional tuning of the prototype and the implementation of some of the suggested improvements would result in significantly better labels. Only four test subjects were used, and larger test should be set up in order to better investigate cluster quality.

The use of additional knowledge bases for clustering has been discussed. It turns out that data from ODP or search engine logs that might benefit traditional clustering approaches are difficult to make use of with the suffix tree clustering used in the prototype.

Although commercial clustering engines exist, clustering is yet to be deployed on major search engines like Google or in Fast ESP. This is presumably because of the computational overhead and because it is so difficult to consistently get high-quality labels for clusters in the noise world of the web. It is possible that clustering will never be used in large-scale web search because it costs too much, but it might be feasable for enterprise search engines that have fewer users.

Bibliography

[1] M. J. Amanda Spink, Dietmar Wolfram and T. Saracevic. Searching the web: The public and their queries. *Journal of the American Society for Information Science and Technology*, 52(3):226–234, 2001.

[2] M. Araújo, G. Navarro, and N. Ziviani. Large text searching allowing errors. In *Proc. of WSP'97*, pages 2–20. Carleton University Press, 1997.

[3] K. W. B. Fung and M. Ester. Large hierarchical document clustering using frequent itemsets. In *SDM03*, 2003.

[4] R. A. Baeza-Yates and B. A. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press / Addison-Wesley, 1999.

[5] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. In *Proceedings of the seventh international conference on World Wide Web*, pages 107–117, 1998.

[6] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2001.

[7] M. H. Craig Silverstein, Hannes Marais and M. Moricz. Analysis of a very large web search engine query log. *SIGIR Forum*, 33(1):6–12, 1999.

[8] D. R. Cutting, J. O. Pedersen, D. Karger, and J. W. Tukey. Scatter/gather: A cluster-based approach to browsing large document collections. In *Proceedings of the Fifteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 318–329, 1992.

[9] P. Ferragina and A. Gulli. A personalized search engine based on web-snippet hierarchical clustering. In *WWW '05: Special interest tracks and posters of the 14th international conference on World Wide Web*, pages 801–810, New York, NY, USA, 2005. ACM Press.

[10] J. Han and M. Kamber. *Data Mining - Concepts and Techniques*. Academic Press, 2001.

[11] H. S. Heaps. *Information Retrieval: Computational and Theoretical Aspects*. Academic Press, Inc., Orlando, FL, USA, 1978.

[12] M. A. Hearst and J. O. Pedersen. Reexamining 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, 1996.

[13] D. A. Hull. Stemming algorithms - a case study for detailed evaluation. *JASIS*, 47(1):70–84, 1996.

[14] D. Koller and M. Sahami. Hierarchically classifying documents using very few words. In D. H. Fisher, editor, *Proceedings of ICML-97, 14th International Conference on Machine Learning*, pages 170–178, Nashville, US, 1997. Morgan Kaufmann Publishers, San Francisco, US.

[15] W. Kraaij and R. Pohlmann. Viewing stemming as recall enhancement. In *Proc. of SIGIR '96*, pages 40–48, 1996.

[16] O. M. Oren Zamir, Oren Etzioni and R. Karp. Fast and intuitive clustering of web documents. In *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining*, pages 287–290, 1997.

[17] S. Osinski and D. Weiss. Conceptual clustering using lingo algorithm: Evaluation on open directory project data. In *IIPWM04*, 2004.

[18] M. F. Porter. An algorithm for suffix stripping. In *Readings in information retrieval*, pages 313–316, 1997.

[19] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Inf. Process. Manage.*, 24(5):513–523, 1988.

[20] E. Ukkonen. Constructing suffix trees on-line in linear time. In *Proceedings of the IFIP 12th World Computer Congress on Algorithms, Software, Architecture - Information Processing '92, Volume 1*, pages 484–492. North-Holland, 1992.

[21] C. J. Van Rijsbergen. *Information Retrieval, 2nd edition*. Dept. of Computer Science, University of Glasgow, 1979.

[22] Webpage. Clusty.com. `http://clusty.com/`. Accessed March 2006.

[23] Webpage. Demonstration of clustering from google. `http://www.searchenginelowdown.com`. Accessed June 2006.

[24] Webpage. Google apis. `http://www.google.com/apis/`. Accessed April 2006.

[25] Webpage. Google directory. `http://directory.google.com/`. Accessed March 2006.

[26] Webpage. List of stopwords. `http://www.idi.ntnu.no/emner/tdt4215/resources/`. Accessed January 2006.

[27] Webpage. Microsoft tests search clustering. `http://www.betanews.com`. Accessed June 2006.

[28] Webpage. Odp/dmoz homepage. `http://dmoz.org/`. Accessed March 2006.

[29] Webpage. Porter stemmer in java. `http://www.tartarus.org/~martin/PorterStemmer`. Accessed January 2006.

[30] Webpage. Spring. `http://www.springframework.com/`. Accessed February 2006.

[31] Webpage. Vivisimo. `http://vivisimo.com/`. Accessed March 2006.

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

[33] D. Weiss and J. Stefanowski. Web search results clustering in polish: Experimental evaluation of carrot. In *IIS03*, 2004.

[34] O. Zamir and O. Etzioni. 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, 1998. ACM Press.

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

APPENDIX A
EVALUATION INSTRUCTIONS AND FORMS

A.1 Instructions

The following instructions (in Norwegian) were given to the test subjects along with the evaluation form:

```
Det er 5 queries (hver har et ark i excel-filen), og jeg viser
top 10 clustere for hvert query. For hvert cluster viser jeg
kandidatlabels og 5 titler fra dokumenter i clusteret.

Deres oppgave er å sette kryss ved det beste kandidatlabelet,
og gi en score for cluster coherence (hvorvidt clusteret det ser
ut til å dreie seg om ett emne) og label quality (hvorvidt det
valgte labelet er et godt navn på et cluster). Poengsummer gis
fra 0-2 der 0 er dårlig, 1 er medium og 2 er god.
```

A.2 Evaluation Form

Figure A.1 shows an example evaluation form for the query 'paris' as it was sent to the test subjects.

A.3 Evaluation Clusters

Figure A.2 and A.3 show the clusters generated by the prototype that were used in the evaluation. For each cluster all candidates are listed with coverage, and the candidates chosen by the prototype as cluster labels are in bold face.

Query: "paris"

Candidate phrase	Coverage	Best label	Coherence	Label quality	Example Document Titles
Cluster 1 (52 documents):					Visit Paris, France - Travel in Paris - Paris Hotels Paris Hotels Guide - Cybevasion France Hotels Hotel Paris - Paris Hotels Reservation - hotels-paris.com Architecture of Paris, France - Great Buildings Online Paris, France Forecast : Weather Underground
France	100 %	X			
Cluster 2 (50 documents):					Paris hotels and Paris city guide with Paris hotel discounts Paris hotel : hotel Paris reservation - Paris hotels discount Visit Paris, France - Travel in Paris - Paris Hotels Paris Hotels Guide - Cybevasion France Hotels Visit Paris in May
Hotels Paris	40 %				
Hotels	100 %				
Paris Hotels	68 %				
Cluster 3 (39 documents):					Paris hotels and Paris city guide with Paris hotel discounts Visit Paris, France - Travel in Paris - Paris Hotels Paris Hotels Guide - Cybevasion France Hotels Information on Paris: Paris Hotels, many monuments such as the Eiffel ... Paris Travel Guide | Fodor's Online
Travel	66 %				
Travel Guide	23 %				
Paris Travel	23 %				
Paris Travel Guide	17 %				
Guide	56 %				
Restaurants	20 %				
Cluster 4 (15 documents):					Visit paris, visit france, tour eiffel, louvre, versailles, mont saint... Fat Tire Bike Tours - Paris, Fat Tyre Bike Tours - Paris :: Enjoy a bi... Paris Museum Pass, Paris Metro Pass, Paris tour and attractions, Cabar... Paris hotels - France - paris apartments rentals and lodging in Paris,... WebMuseum: Paris: Tours
Tour	100 %	X			
Cluster 5 (14 documents):					Paris hotels and Paris city guide with Paris hotel discounts Paris Hotels - Find Hotel Deals for Paris Hotels Paris, France Forecast : Weather Underground Paris Digest, the Paris city guide and online portal Paris hotels, discount, budget and luxury accommodation. City hotels...
City	100 %	X			
Cluster 6 (14 documents):					Paris Net - Apartments for rent in Paris Vacation apartment rentals PARIS APARTMENTS Paris Apartment HOTELS PARIS, accommodation rentals Paris Apartments : Apartment & Flat Rentals in Paris Paristay apartment rentals : authentic apartments for rent in Paris fr... Paris Apartments in Paris Apartment Rentals RENTALS IN PARIS
Rent	35 %				
Paris Apartments	78 %				
Rentals	85 %				
Apartments	92 %				
Apartment Rentals	50 %				
Paris Apartment Rentals	50 %				
Cluster 7 (13 documents):					Paris Information Paris hotel : hotel Paris reservation - Paris hotels discount Information on Paris: Paris Hotels, many monuments such as the Eiffel ... Paris for Visitors - Travel and Tourist Information Paris tourist information - VIRTOURIST.COM
Information	100 %	X			
Cluster 8 (12 documents):					Hotel Paris - Paris Hotels Reservation - hotels-paris.com Hotels in paris - Hotel paris Left bank Latin Quarter st Germain. Char... Home page, france, travel france, france paris, paris france, france ... WebFrance International, Paris, France Paris France WebFrance International
Paris France	100 %	X			
Cluster 9 (11 documents):					Paris hotel : hotel Paris reservation - Paris hotels discount Hotel Paris - Paris Hotels Reservation - hotels-paris.com Paris Hotel reservation service offers top quality charming Paris hote... Hotels: Paris, France. Hotel Accommodation. WWW.OUR-PARIS.COM Paris Hotels & Lodging - Discount Paris Hotel Reservations - Discount ...
Paris Hotel Reservation	36 %				
Hotel Reservation	63 %				
Reservation	100 %				
Cluster 10 (11 documents):					Club Paris - Paris Hilton Downtown Orlando Florida Night... Paris Hilton Pictures AskMen.com - Paris Hilton Paris Hilton Pictures, Biography, Filmography, News, Videos, Wallpaper... Paris Hilton, Paris Hilton picture gallery, free wallpapers, photo gal...
Hilton	100 %				
Paris Hilton	90 %				

Figure A.1: Evaluation Form

hollywood

Candidate phrase	Coverage
Cluster 1 (32 documents):	
Los Angeles	65 %
Los	68 %
California	62 %
Angeles	68 %
Cluster 2 (17 documents):	
Movies	100 %
Cluster 3 (15 documents):	
Hotels	100 %
Hollywood Hotels	53 %
Cluster 4 (14 documents):	
Film	100 %
Cluster 5 (11 documents):	
Store	63 %
Directory	63 %
Cluster 6 (10 documents):	
City	70 %
Florida	60 %
Cluster 7 (10 documents):	
West Hollywood	100 %
Cluster 8 (10 documents):	
Books	100 %
Cluster 9 (10 documents):	
Real Estate	70 %
Jobs	60 %
Estate	70 %
Cluster 10 (9 documents):	
Entertainment	100 %

paris

Candidate phrase	Coverage
Cluster 1 (52 documents):	
France	100 %
Cluster 2 (50 documents):	
Hotels Paris	40 %
Hotels	100 %
Paris Hotels	68 %
Cluster 3 (39 documents):	
Travel	66 %
Travel Guide	23 %
Paris Travel	23 %
Paris Travel Guide	17 %
Guide	56 %
Restaurants	20 %
Cluster 4 (15 documents):	
Tour	100 %
Cluster 5 (14 documents):	
City	100 %
Cluster 6 (14 documents):	
Rent	35 %
Paris Apartments	78 %
Rentals	85 %
Apartments	92 %
Apartment Rentals	50 %
Paris Apartment Rentals	50 %
Cluster 7 (13 documents):	
Information	100 %
Cluster 8 (12 documents):	
Paris France	100 %
Cluster 9 (11 documents):	
Paris Hotel Reservation	36 %
Hotel Reservation	63 %
Reservation	100 %
Cluster 10 (11 documents):	
Hilton	100 %
Paris Hilton	90 %

jaguar

Candidate phrase	Coverage
Cluster 1 (57 documents):	
Cars	100 %
Jaguar Cars	52 %
Cluster 2 (37 documents):	
Cats	100 %
Cluster 3 (26 documents):	
Panthera Onca	88 %
Panthera	96 %
Onca	96 %
Cluster 4 (22 documents):	
Parts	68 %
Jaguar Parts	45 %
Car Parts	9 %
View Cart	13 %
View	27 %
Jaguar Auto Parts	22 %
Auto Parts	31 %
Oem Parts	9 %
Cart	22 %
Jaguar Oem Parts	9 %
Cluster 5 (20 documents):	
Reviews	100 %
Cluster 6 (19 documents):	
Jaguar North America Joins Happy Hollow Park	10 %
America	42 %
Central	21 %
North America Joins Happy Hollow Park	10 %
America Joins Happy Hollow Park	10 %
Join	21 %
Central America	10 %
Zoo	47 %
South America	21 %
South	31 %
Cluster 7 (19 documents):	
Auto	100 %
Cluster 8 (18 documents):	
Service	44 %
Vehicles	61 %
Owners	50 %
Finances	50 %
Company	33 %
Cluster 9 (17 documents):	
Type	100 %
Cluster 10 (16 documents):	
Buy	100 %

Figure A.2: Evaluation Clusters

red hot chili peppers

Candidate phrase	Coverage
Cluster 1 (54 documents):	
Music	100 %
Cluster 2 (48 documents):	
Artists	79 %
Albums	56 %
Cluster 3 (36 documents):	
Hot Chili Peppers Tickets	61 %
Hot Chili Peppers Concert Tickets	22 %
Red Hot Chili Peppers Concert Tickets	22 %
Concert Tickets	30 %
Chili Peppers Tickets	63 %
Peppers Tickets	63 %
Red Hot Chili Peppers Tickets	61 %
Tickets	91 %
Chili Peppers Concert Tickets	22 %
Peppers Concert Tickets	22 %
Cluster 4 (18 documents):	
Lyrics	100 %
Chili Peppers Lyrics	61 %
Red Hot Chili Peppers Lyrics	61 %
Peppers Lyrics	61 %
Hot Chili Peppers Lyrics	61 %
Cluster 5 (18 documents):	
Songs	100 %
Cluster 6 (18 documents):	
Videos	100 %
Cluster 7 (16 documents):	
Review	100 %
Cluster 8 (16 documents):	
Dvds	68 %
Book	62 %
Cluster 9 (15 documents):	
Stadium Arcadium	86 %
Arcadium	86 %
Stadium	100 %
Cluster 10 (12 documents):	
Downloads	100 %

apple

Candidate phrase	Coverage
Cluster 1 (44 documents):	
Items Ebay	15 %
Apple Macintosh Computers Items Ebay	4 %
Macintosh Computers	13 %
Low Prices	13 %
Map	38 %
Items Low Prices	13 %
Items	34 %
Macintosh Computers Items Ebay	4 %
Pay	56 %
Macintosh	34 %
Cluster 2 (41 documents):	
Developers	21 %
Faq	19 %
Pay Developers	9 %
Linux	17 %
Zdnet	21 %
Blogs	17 %
Hardware	17 %
Stories	17 %
Books	19 %
White Papers	12 %
Cluster 3 (28 documents):	
Consumer Electronics	7 %
Electronics	14 %
Player Accessories	14 %
Ipod	92 %
Apple Ipod	46 %
Accessories Items	7 %
Accessories	32 %
Players	28 %
Cluster 4 (24 documents):	
Computer	100 %
Apple Computer	62 %
Cluster 5 (19 documents):	
Reviews	100 %
Cluster 6 (18 documents):	
Apple Store Online	16 %
Store Online	16 %
Online	33 %
Apple Store	38 %
Visit	33 %
Store	83 %
Cluster 7 (18 documents):	
Mac	100 %
Cluster 8 (17 documents):	
Read	23 %
Cars	23 %
Airport Express	11 %
Compare Prices Ciao	11 %
Read Reviews	11 %
Airport	29 %
Desktops	23 %
Apple Airport	23 %
Online Stores	11 %
Compare Prices	23 %

Candidate phrase	Coverage
Cluster 9 (10 documents):	
Support	100 %
Cluster 10 (9 documents):	
Software	100 %

Figure A.3: Evaluation Clusters

APPENDIX B
TEST RESULTS

This chapter lists the results of the user survey for all five queries. The three first columns show the points given by each of the four test subjects (labelled S1-S4) for precision, coherence and label quality for each of the ten clusters per query. The last column shows the average values (averaged over all test subjects).

hollywood

Precision				Coherence				Label quality				Cluster	Precision	Coherence	Quality
S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4				
1	1	1	1	2	1	2	2	1	1	2	2	1	1	1,75	1,5
1	1	1	1	2	2	2	2	2	0	2	2	2	1	2	1,5
1	0,5	1	0,5	2	1	2	2	2	1	2	2	3	0,75	1,75	1,75
1	1	1	1	2	2	2	2	2	1	2	2	4	1	2	1,75
1	1	1	0	1	0	1	1	0	0	1	1	5	0,75	0,75	0,5
0	0	0	1	0	1	0	1	0	1	1	0	6	0,25	0,5	0,5
1	1	1	1	2	2	2	2	0	1	1	2	7	1	2	1
1	1	1	1	2	2	2	1	1	0	2	1	8	1	1,75	1
1	1	1	1	1	1	1	1	1	1	2	1	9	1	1	1,25
1	1	1	1	2	2	2	2	2	0	2	2	10	1	2	1,5
												All	0,875	1,55	1,225

paris

Precision				Coherence				Label quality				Cluster	Treffrate	Coherence	Quality
S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4				
1	1	1	1	2	2	2	2	0	0	2	2	1	1	2	1
1	0,5	1	0,5	2	2	2	2	2	2	2	2	2	0,75	2	2
1	0,5	1	0,5	1	1	2	2	1	1	2	2	3	0,75	1,5	1,5
1	1	1	1	2	2	2	2	0	0	2	2	4	1	2	1
1	1	1	1	2	2	2	2	0	0	1	2	5	1	2	0,75
0	0,5	0	0,5	2	1	2	2	2	1	2	2	6	0,25	1,75	1,75
1	1	1	1	2	2	2	1	1	0	1	1	7	1	1,75	0,75
1	1	1	1	2	2	2	2	0	0	1	2	8	1	2	0,75
1	0,5	1	0,5	2	1	2	2	2	1	2	2	9	0,75	1,75	1,75
1	1	1	1	2	2	2	1	2	2	2	2	10	1	1,75	2
												All	0,85	1,85	1,325

Figure B.1: Evaluation Results

jaguar

Precision				Coherence				Label quality				Cluster	Treffrate	Coherence	Quality
S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4				
1	0,5	1	0,5	2	1	2	2	2	1	2	2	1	0,75	1,75	1,75
1	1	1	1	2	2	2	2	2	0	2	2	2	1	2	1,5
1	0	0	1	2	0	2	1	2	0	2	1	3	0,5	1,25	1,25
1	0	1	0,5	2	1	1	2	1	2	2	2	4	0,625	1,5	1,75
1	1	0	1	2	2	2	2	1	1	2	1	5	0,75	2	1,25
0	0	0	0	0	0	0	0	2	0	1	0	6	0	0	0,75
1	1	1	1	2	2	2	2	2	0	2	2	7	1	2	1,5
1	1	1	1	0	0	1	2	1	0	1	1	8	1	0,75	0,75
1	1	1	1	2	2	2	1	0	0	1	0	9	1	1,75	0,25
1	1	1	1	2	2	2	1	0	0	1	1	10	1	1,75	0,5
												All	0,7625	1,475	1,125

apple

Precision				Coherence				Label quality				Cluster	Treffrate	Coherence	Quality
S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4				
0	0	0	0	1	0	1	2	1	1	1	2	1	0	1	1,25
0	0	0	0	0	0	0	0	1	0	1	1	2	0	0	0,75
1	0,5	0	1	1	1	0	2	2	1	1	2	3	0,625	1	1,5
1	0,5	1	0,5	2	1	2	2	2	1	2	2	4	0,75	1,75	1,75
1	1	1	1	2	2	2	2	1	0	2	1	5	1	2	1
0,5	0	1	0	1	1	2	2	1	1	1	2	6	0,375	1,5	1,25
1	1	1	1	2	2	2	2	1	2	2	2	7	1	2	1,75
0	0	0	0	0	0	0	1	2	1	1	1	8	0	0,25	1,25
1	1	1	1	2	2	2	1	1	1	2	1	9	1	1,75	1,25
1	1	1	1	2	2	2	1	2	0	2	1	10	1	1,75	1,25
												All	0,575	1,3	1,3

red hot chili peppers

Precision				Coherence				Label quality				Cluster	Treffrate	Coherence	Quality
S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4				
1	1	1	1	2	2	2	2	2	1	2	2	1	1	2	1,75
1	1	0	1	0	1	1	2	1	0	2	2	2	0,75	1	1,25
0	0	1	0,5	2	2	2	2	2	2	2	2	3	0,375	2	2
1	0,5	1	0,5	2	1	2	2	2	2	2	2	4	0,75	1,75	2
1	1	1	1	2	2	2	2	2	0	2	2	5	1	2	1,5
1	1	1	1	2	2	2	2	2	0	2	2	6	1	2	1,5
1	1	1	1	2	2	2	2	1	0	2	2	7	1	2	1,25
1	1	0	1	0	1	1	2	1	0	1	2	8	0,75	1	1
1	1	1	1	2	1	1	2	1	1	2	1	9	1	1,5	1,25
1	1	1	1	2	2	2	1	2	0	2	2	10	1	1,75	1,5
												All	0,8625	1,7	1,5

Figure B.2: Evaluation Results

APPENDIX C
STOPWORDS

The prototype uses a standard list of stopwords, and an additional list of web specific stop words. The following words are the web spescific stopwords:

```
home
search
login
user
sign
amp
page
site
find
contact
free
top
mail
version
web
log
password
username
register
navigation
```

The following are the stopwords used in the prototype, acquired from [26].

```
a
a's
able
about
above
according
accordingly
across
actually
after
afterwards
again
against
ain't
all
allow
allows
almost
alone
along
already
also
although
always
am
among
amongst
an
and
another
any
anybody
anyhow
anyone
anything
anyway
anyways
anywhere
apart
appear
appreciate
appropriate
are
aren't
around
as
aside
ask
asking
associated
at
available
away
awfully
b
be
became
because
become
becomes
becoming
been
before
beforehand
behind
being
believe
below
beside
besides
best
better
between
beyond
both
brief
but
by
c
c'mon
c's
came
can
can't
cannot
cant
cause
causes
certain
certainly
changes
clearly
```

```
co
com
come
comes
concerning
consequently
consider
considering
contain
containing
contains
corresponding
could
couldn't
course
currently
d
definitely
described
despite
did
didn't
different
do
does
doesn't
doing
don't
done
down
downwards
during
e
each
edu
eg
eight
either
else
elsewhere
enough
entirely
especially
et
etc
even
ever
every
everybody
everyone
everything
everywhere
ex
exactly
example
except
f
far
few
fifth
first
five
followed
following
follows
for
former
formerly
forth
four
from
further
furthermore
g
get
gets
getting
given
gives
go
goes
going
gone
got
gotten
greetings
h
had
hadn't
happens
hardly
has
hasn't
have
haven't
having
he
he's
hello
help
hence
her
here
here's
hereafter
hereby
herein
hereupon
hers
herself
hi
him
himself
his
hither
hopefully
how
howbeit
however
i
i'd
i'll
i'm
i've
ie
if
ignored
immediate
in
inasmuch
inc
indeed
indicate
indicated
indicates
inner
insofar
instead
into
inward
is
isn't
it
it'd
it'll
it's
its
itself
j
just
k
keep
keeps
kept
know
knows
known
l
last
lately
later
latter
latterly
least
less
lest
let
let's
like
liked
likely
little
look
looking
looks
ltd
m
mainly
many
may
maybe
me
mean
meanwhile
merely
might
more
moreover
most
mostly
much
must
my
myself
n
name
```

namely
nd
near
nearly
necessary
need
needs
neither
never
nevertheless
new
next
nine
no
nobody
non
none
noone
nor
normally
not
nothing
novel
now
nowhere
o
obviously
of
off
often
oh
ok
okay
old
on
once
one
ones
only
onto
or
other
others
otherwise
ought
our
ours
ourselves
out
outside
over
overall
own
p
particular
particularly
per
perhaps
placed
please
plus
possible
presumably
probably
provides
q
que
quite
qv
r
rather
rd
re
really
reasonably
regarding
regardless
regards
relatively
respectively
right
s
said
same
saw
say
saying
says
second
secondly
see
seeing
seem
seemed
seeming
seems
seen
self
selves
sensible
sent
serious
seriously
seven
several
shall
she
should
shouldn’t
since
six
so
some
somebody
somehow
someone
something
sometime
sometimes
somewhat
somewhere
soon
sorry
specified
specify
specifying
still
sub
such
sup
sure
t
t’s
take
taken
tell
tends
th
than
thank
thanks
thanx
that
that’s
thats
the
their
theirs
them
themselves
then
thence
there
there’s
thereafter
thereby
therefore
therein
theres
thereupon
these
they
they’d
they’ll
they’re
they’ve
think
third
this
thorough
thoroughly
those
though
three
through
throughout
thru
thus
to
together
too
took
toward
towards
tried
tries
truly
try
trying
twice
two
u
un
under
unfortunately
unless

unlikely
until
unto
up
upon
us
use
used
useful
uses
using
usually
uucp
v
value
various
very
via
viz
vs
w
want
wants
was
wasn't
way
we
we'd
we'll
we're
we've
welcome
well
went
were
weren't
what
what's
whatever
when
whence
whenever
where
where's
whereafter
whereas
whereby
wherein
whereupon
wherever
whether
which
while
whither
who
who's
whoever
whole
whom
whose
why
will
willing
wish
with
within
without
won't
wonder
would
would
wouldn't
x
y
yes
yet
you
you'd
you'll
you're
you've
your
yours
yourself
yourselves
z
zero