Clustering Web Documents:
A Phrase-Based Method for Grouping Search Engine Results

Oren Eli Zamir

A dissertation submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

University of Washington

1999

Program Authorized to Offer Degree: Computer Science & Engineering

University of Washington
Graduate School

This is to certify that I have examined this copy of a doctoral dissertation by

Oren Eli Zamir

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examination committee have been made.

Chair of Supervisory Committee:

Oren Etzioni

Reading Committee:

Dan Weld

Susan Dumais

Date:

In presenting this dissertation in partial fulfillment of the requirements for the Doctorial degree at the University of Washington, I agree that the Library shall make its copies freely available for inspection. I further agree that extensive copying of this thesis is allowable only for scholarly purposes, consistent with ``fair use'' as prescribed in the U.S. Copyright Law. Requests for copying or reproduction of this dissertation may be referred to University Microfilms, 1490 Eisenhower Place, P.O. Box 975, Ann Arbor, MI 48106, to whom the author has granted “the right to reproduce and sell (a) copies of the manuscript in microform and/or (b) printed copies of the manuscript made from microform.”

Signature ___________________________

Date _______________________________

University of Washington

Abstract

Clustering Web Documents:

A Phrase-Based Method for Grouping Search Engine Results

Oren Eli Zamir

Chair of Supervisory Committee

Associate Professor Oren Etzioni

Department of Computer Science & Engineering

This dissertation investigates whether the automatic grouping of similar documents (document clustering) is a feasible method of presenting the results of Web search engines. We identify several key requirements for document clustering of search engine results: clustering quality, concise and accurate cluster descriptions, and speed. In response, we present a novel clustering algorithm – Suffix Tree Clustering (STC) specifically designed for this task in several respects. First, STC groups documents based on shared phrases. Second, it allows overlapping clusters. Finally, STC is a fast, incremental, and linear-time (in the number of documents) algorithm. We have evaluated the clustering quality of STC and showed it superior to other commonly used algorithms on Web search engine results. We have also shown STC to be significantly faster than other linear-time algorithms when clustering search engine snippets. We have incorporated STC into the Grouper system, which is available at *http://www.cs.washington.edu/research/clustering*. Grouper is, to our knowledge, the first implementation of a post-retrieval document-clustering interface to a Web search engine

Inspired by STC's success, we studied the use of phrases for clustering across a variety of clustering algorithms. Phrases contain more information than single words (information regarding proximity and order of words) and have the equally important advantage of having a higher descriptive power. We have shown that using phrases substantially improves the performance of all algorithms tested including hierarchical group-average, k-means, single pass, Fractionation and Buckshot.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Number Page

ACKNOWLEDGMENTS

I owe many thanks to a number of people who have supported, inspired, motivated, challenged and entertained me throughout my graduate studies. First and foremost, I would like to thank my advisor Oren Etzioni. Oren taught me how to do research, to write and to talk in public. He was also a good friend, helping me stay motivated and focused throughout this lengthy period. Thanks also go to the other members of my committee: Dan Weld and Susan Dumais for many valuable insights and advice.

I would like to thank Erik Selberg for allowing me to use the HuskySearch code and for his collaboration and many helpful tips. Thanks also go to Zhanya Sigal and Mellisa Johnson for their help in implementing Grouper. I would also like to thank my friends and colleagues at the University of Washington: Mike Perkowitz, Marc Friedman, Neal Lesh, Tessa Lau, Adam Carlson, Jonathan Shakes, Marc Langheinrich, Omid Madani, Frederic Pighin, Sujay Parekh, Marcus Mock, Matthai Philipose, Todd Millstein and Greg Badros. I would especially want to thanks AnHai Doan, my officemate and friend, who has always been there for me. Not only was AnHai a great source of advice, he also helped make my time in Seattle engaging and memorable.

I would like to thanks my family, my mother Ora, my father Ehud, and my two sisters Ruth and Yael for all their support and encouragement these many years. Finally, I would like to thank Simona who shared every nuance of the last five years with me. Her love and support has given me the strength and confidence to complete this endeavor.

DEDICATION

In memory of Alon (“gingi”) Cohen, a good friend who enthusiastically encouraged me to go to graduate school but did not make it to my graduation day.

Chapter 1

INTRODUCTION

1.1 Motivation

The amount of available information on the Web is increasing rapidly. The publicly indexable Web contains an estimated 800 million pages as of February 1999, encompassing about 15 terabytes of information or about 6 terabytes of text after removing HTML tags, comments, and extra whitespace [Lawrence and Giles, 99]. As noted by Lawrence and Giles, "the revolution that the Web has brought to information access is not so much due to the availability of information (huge amounts of information has long been available in libraries and elsewhere), but rather the increased efficiency of accessing information, which can make previously impractical tasks practical". But how can users find the information they are seeking in such an unorganized, unstructured and decentralized place?

As of December 1998, 85% of Web users used search services to locate Web pages and 60% used Web directories [Kehoe and Pitkow, 99]. On the other hand, in the same survey 45% of the users stated that one of the biggest problems of using the Web was the inability to find the information they were looking for. There are some well-studied reasons why finding information using Web search engines is not always successful. No single search engine has indexed the entire Web. As of August 1999, FAST reports to have indexed the largest number of pages – 200 million (less than 25% of the estimated size of the indexable Web), while an independent study showed no search engine to index more than 16% of the Web [Lawrence and Giles, 99]. Another factor that decreases search engine usefulness is the dynamic nature of the Web, resulting in many "dead links" and "out of date" pages that have changed since indexed. But even excepting these factors, finding relevant information using Web search engines often fails.

Document retrieval systems typically present search results in a ranked list, ordered by their estimated relevance to the query. The relevancy is estimated based on the similarity between the text of a document and the query. Such ranking schemes work well when users can formulate a well-defined query for their searches. However, users of Web search engines often formulate very short queries (70% are single word queries [Motro, 98]) that often retrieve large numbers of documents. Based on such a condensed representation of the users' search interests, it is impossible for the search engine to identify the specific documents that are of interest to the users. Moreover, many webmasters now actively work to influence rankings. These problems are exacerbated when the users are unfamiliar with the topic they are querying about, when they are novices at performing searches, or when the search engine's database contains a large number of documents. All these conditions commonly exist for Web search engine users. Therefore the vast majority of the retrieved documents are often of no interest to the user; such searches are termed *low precision searches*.

The low precision of the Web search engines coupled with the ranked list presentation force users to sift through a large number of documents and make it hard for them to find the information they are looking for. As low precision Web searches are inevitable, tools must be provided to help users "cope" with (and make use of) these large document sets. Such tools should include means to easily browse through large sets of retrieved documents.

The motivation for this research is to make search engine results easy to browse. Document clustering algorithms attempt to group similar documents together. Clustering the results of Web search engines can provide a powerful browsing tool. One of the central questions asked in this dissertation is:

> **Is the automatic grouping of similar documents (document clustering) a feasible method of presenting the results of Web search engines?**

Document clustering has initially been investigated in Information Retrieval mainly as a means of improving the performance of search engines by pre-clustering the entire corpus [Jardine and van Rijsbergen, 71]. The *cluster hypothesis* [van Rijsbergen, 79] stated that similar documents will tend to be relevant to the same queries, thus the automatic detection of clusters of similar documents can improve recall by effectively broadening a search request. However we are investigating clustering as a means of browsing large retrieved document sets. We therefore need to slightly modify the cluster hypothesis to suit this domain. Our modified *user-cluster hypothesis* is that users have a mental model of the topics and subtopics of the documents present in the result set; similar documents will tend to belong to the same mental category in the users' model. Thus the automatic detection of clusters of similar documents can help the user in browsing the result set.

Clustering of search results can help users in three ways: (1) it can allow them to find the information they are looking for more easily, (2) it can help them to realize faster that a query is poorly formulated (*e.g.*, too general) and to reformulate it, and (3) it can reduces the fraction of the queries on which the user gives up before reaching the desired information. For example, if a user wishes to find salsa recipes on the Web, and performs a search using the query "salsa", only 10% of the returned documents will be related to salsa recipes (the rest will relate to salsa music, salsa products that can be bought on the Web and a software product called "salsa"; many documents will have no apparent connection to salsa at all). If we were to cluster the results, the user could find the cluster relating to salsa recipes and thus save valuable browsing time.

We have identified some key requirements for document clustering of search engine results:

1. Coherent Clusters: The clustering algorithm should group similar documents together.
2. Efficiently Browsable: The user needs to determine at a glance whether the contents of a cluster are of interest. Therefore, the system has to provide concise and accurate cluster descriptions.
3. Speed: The clustering system should not introduce a substantial delay before displaying the results.

In preliminary experimentation carried out at the beginning of this study we found Web documents, and especially search engine snippets, to be poor candidates for clustering because they are short and often poorly formatted. This led us to consider the use of phrases in the clustering of search engine results, as they contain more information than simple words (information regarding proximity and order of words). Phrases have the equally important advantage of having a higher descriptive power (compared to single words). This is very important when attempting to describe the contents of a cluster to the user in a concise manner. This leads us to the second major questions of this dissertation:

> **Will the use of phrases help in achieving high quality groupings of search engine results? Can phrase-based clustering be done quickly?**

1.2 Overview

The basic question asked in this dissertation is: ”Is the automatic grouping of similar documents a feasible method of presenting the results of Web search engines?” By feasible we in fact mean three things: (1) Can the automatic grouping of documents actually cluster similar document together (we call this the “clustering quality”)? (2) Can it be done fast enough, *i.e.*, with little perceptible delay to the user? (3) Will it in fact be useful to the user?

The clustering quality and the speed are both influenced by the clustering algorithm that is used and by the characteristics of the documents that are clustered (Web search results). Therefore the first two questions can be investigated via offline evaluations.

The third question is harder to investigate. In order to evaluate usefulness to users we implemented and deployed a clustering interface to Web searches. This allowed us to obtain quantitative evaluations via analysis of the system’s logs and a user study as well as qualitative impressions via user feedback.

1.2.1 The Suffix Tree Clustering Algorithm

The core of a clustering interface to search results is the clustering algorithm that is used. We started this research by trying to use a set of well-known clustering algorithms (presented in chapter 2) and were dissatisfied with the results. We hypothesized that there were two main reasons for this failure. First, Web documents, and especially search engine snippets, are short and often poorly formatted, thus they are difficult to cluster. Second, the algorithms that were sufficiently fast were all model-based algorithms. Model-based algorithms have a priori assumptions as to the model describing the data (*e.g.*, the k-means algorithm assumes spherical and equal-sized clusters). These algorithms search for the most probable model parameters, given the data and the a priori assumptions regarding the model. When the data does not fit the model these algorithms perform poorly. There is no reason to believe that Web documents fit these models. These insights lead us to design a novel clustering algorithm – Suffix Tree Clustering (STC) – with two key features: the use of phrases and a simple cluster definition that does not assume a specific model for the data.

STC has two main steps. In the first step it searches for all sets of documents that share a common phrase. We call these sets *phrase clusters* and they are found using the suffix tree data structure. In the second step we merge these phrase clusters into clusters (appropriately, we call these *merged clusters*). We merge two phrase clusters based on the percent of the documents that contain both phrases.

Several characteristics make STC a promising candidate for the clustering of search results. First, it is phrase-based, generating clusters by grouping documents that share many phrases. Phrases are also useful in constructing concise and accurate descriptions of the clusters. Second, It does not adhere to any model of the data. Its only assumption is that documents on the same topic will share common phrases. Third, STC allows overlapping clusters. It is important to avoid confining each document to only one cluster since documents often have multiple topics, and thus might be similar to more than one group of documents. Fourth, STC uses a simple cluster definition – all documents containing one of the cluster's phrases are members of the cluster. We have noticed in our experiments that users have a hard time interpreting the clusters when the clustering algorithm has a complex and non-intuitive definition of a cluster. A simple cluster definition will help users understand precisely which documents are present in or absent from a cluster. Finally, STC is a fast incremental, linear time (in the number of documents) algorithm, which makes it suitable for online clustering of Web searches.

1.2.2 Offline Evaluation of the Quality of Clustering Algorithms

Can the automatic grouping of documents actually cluster similar documents together? To address this question we wanted to evaluate the clustering quality of the STC algorithm and compare it to other commonly used clustering algorithms. There were three sub-questions we wanted to address:

1. What is the clustering quality of the STC algorithm and how does it compare with other clustering algorithm?
2. Which of STC's features contribute to its high quality clusters? Is it the use of phrases? Is it overlapping clusters?
3. Will phrases also improve the clustering quality of other clustering algorithms besides STC?

We used three document collections is our experiment: two Web Search Results (WSR) collections that we constructed for this research (one of search engine snippets and the other of their corresponding documents), and a standard medical collection (OHSUMED) that is used extensively in IR research. The clustering quality was compared using two quality-evaluation approaches: the IR approach and the "merge-then-cluster" approach. The consistency of the results across the different evaluation metrics and text collections was encouraging and added to our confidence in the results.

Our experiments demonstrated that STC and k-means produce the highest quality clusters. Of these, STC was superior on the Web collections, while k-means was slightly better on the OHSUMED collection.

We proceeded to investigate the contribution of STC's features to its success. Was it that STC allows overlapping clusters? We have shown that when overlapping clusters are not allowed, the quality of the STC clusters decreases by more than 20%. We therefore asked could overlapping clusters improve the performance of other clustering algorithms as well? When we adapted the k-means algorithm to allow overlapping clusters we observed a 7% increase in its performance (it was still inferior to STC), suggesting that other clustering algorithms could also benefit from this capability.

STC uses phrases to detect similarity between documents. Would its clustering quality decrease without the use of phrases? Our experiment showed that without the use of multiword phrases, the quality of the clusters created by STC fell by close to 20%.

After showing the advantages of using phrases in STC, we turned our attention to other clustering algorithms. Can phrases improve the performance of vector-based clustering algorithm as well? We used a suffix tree to identify phrases in the document set and added these as attributes to the documents' vector representations. We were able to show substantial improvement in the performance of all vector-based algorithms tested. The improvement was most noticeable in the algorithms that performed poorly using words alone. We believe this is because using phrases reduces the amount of noise in the data, as chance similarities are less likely to occur.

The STC algorithm identifies phrases using a suffix tree, however alternative phrase generation methods have been explored in the IR literature. These include syntactic phrases (such as the identification of compound nouns using natural language processing techniques; these are too slow for our application) as well as statistical phrases such n-grams – phrases created by using sliding windows of *n* words over the text (a suffix tree is also a statistical technique). We wanted to explore whether STC was benefiting from using the long phrases that the suffix tree identifies: Would there be a significant difference in performance between the use of suffix tree phrases compared to n-grams? We showed that there is only a minor advantage to using suffix tree phrases over using 2-grams. This means that the use of long phrases (3 words or more in length) does not add much to the quality of the results. We believe that the main advantage of using suffix tree phrases lies in their descriptive power – longer phrases are much better at describing the contents of a cluster to the user.

We also wanted to understand why phrases are better than words: Does the advantage of using phrases lie in the information present in the adjacency and order of the words, or simply in their being multiword features? To address these issues we generalized the STC algorithm such that phrase clusters are defined using *frequent sets* – sets of documents that share a set of words. We call this version of the STC algorithm Frequent Set Clustering (FSC). We were able to show that the quality of FSC clusters is comparable to that of STC on the short documents of WSR-SNIP. However, in the WSR-DOCS collection and even more so in the OHSUMED collection, STC outperformed FSC. We conclude that the information present in the order and adjacency of words is important and does contribute to the clustering quality.

1.2.3 Evaluation of Clustering Speed

Is the STC algorithm fast? How does its speed compare with other clustering algorithms? Does the use of phrases increase the clustering time and render online clustering infeasible? After investigating the clustering quality of STC, we needed to evaluate its speed. We compared STC to other linear-time clustering algorithms and showed STC to be significantly faster than k-means (the fastest of the vector-based algorithms) when clustering search engine snippets. When clustering Web documents the speed advantage of STC disappears (both STC and k-means are linear with respect to the average document length, but this constant is larger for STC).

In our client-clustering model, clustering of search engine snippets can be performed much faster than clustering their corresponding Web documents. Our clustering algorithm must therefore be *snippet tolerant* – it ought to produce high quality clusters even when it only has access to the snippets returned by the search engines – as most users are unwilling to wait while the system downloads the original documents off the Web. But is snippet clustering a reasonable approximation to document clustering?

We compared the performance of clustering algorithms on collections of search engine snippets and on their corresponding Web documents. The quality of STC clusters on snippets was some 20% lower than on Web documents. We believe this to be a moderate decrease in quality that shows that snippet clustering is a reasonable approximation to document clustering. This moderate decrease in quality is surprising as a Web document contained approximately 10 times more words on average than a snippet. One explanation is that the snippets represent (partially successful) attempts by the search engines to extract meaningful words and phrases from the original documents.

1.2.4 Grouper – Fielding a Clustering Interface to Search Results

We have shown the STC algorithm to be fast and to produce high quality clusters. The STC algorithm can therefore be used to organize Web search results into clusters. But will such a system be in fact useful to the users? And if so, for what information tasks is it most suited? In order to evaluate these questions we have implemented and deployed a clustering interface to Web searches. This allowed us to obtain quantitative evaluations via analysis of the system's logs and a user study as well as qualitative impressions via user feedback.

Grouper is, to our knowledge, the first implementation of a post-retrieval document-clustering interface to a Web search engine. It is integrated into the HuskySearch meta search service as one of three possible interface modes, the others being a ranked list and a presentation sorted by sites. HuskySearch retrieves results from several popular Web search engines, and Grouper clusters the results using the STC algorithm. Grouper is publicly available at http://www.cs.washington.edu/research/clustering.

Figure 1.1 presents the main results page of Grouper I (Grouper II will be described shortly) as a result of the query "clinton" (performed on May 4th 1999), which retrieved 298 documents. In the interest of clarity, only the first three clusters are shown. The first cluster in this example, containing 37 documents, appears to contain documents pertaining to the Clinton-Lewinsky scandal and is characterized by four phrases: “Monica Lewinsky”, “Clinton’s scandals”, “Kenneth Starr investigation” and “Hillary Clinton”. The second and third clusters appear to relate to the presidential candidacy of vice president Al Gore and to the Paula Jones scandal, respectively.

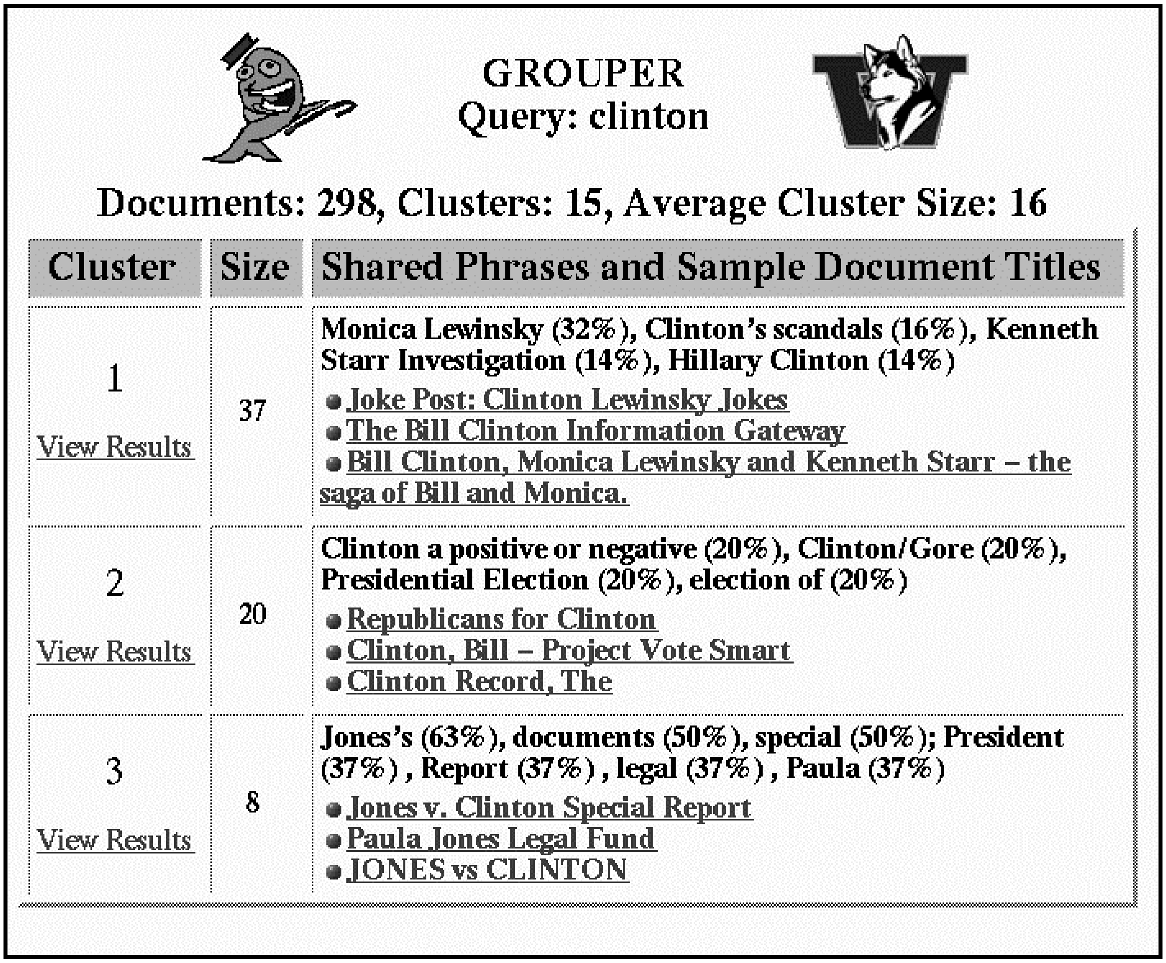


Figure 1.1: Grouper I’s clusters interface

The main results page of Grouper for the query “clinton”. Each row in the table is the *summary* of a cluster – an attempt to convey the content of the documents in the cluster. It includes the size of the cluster, *shared phrases* – phrases that appear in many documents of the cluster, and up to three *sample titles* of documents in the cluster. The numbers appearing in parenthesis after each phrase indicate the percentage of the documents in the cluster that contain the phrase. In the example above only the first three clusters are shown.

As mentioned previously, it is not enough for a clustering system to create coherent clusters, but the system must also convey the contents of the clusters to the users concisely and accurately. In Grouper I we applied several techniques to address this issue. As STC creates clusters based on phrases that are shared by many of their documents, Grouper uses these phrases to describe the cluster. We have found these phrases to be extremely useful in characterizing the cluster's content. Since each cluster can contain many phrases, and our goal is to create a compact cluster summary, we display at most the six highest scoring phrases. We have found that in many cases some of the displayed phrases are quite similar and therefore redundant. We have developed a set of three heuristics that help us to identify (and exclude) redundant phrases. We also display frequent words that appear in the cluster but do not appear in its phrases. The documents of a cluster are sorted based on the number of the cluster's phrases each contains as this results in more informative cluster summaries.

By fielding Grouper I on the Web, we were able to identify some of its shortcomings: the results were, at times, confusing, not always useful, and not always intuitive for the novice user. Sometimes, Grouper I identifies phrases that are common in the collection but are not interesting to the user (*e.g.*, "This was last updated on"). When they are used to construct clusters the result might be random and useless clusters. Even the merging of meaningful phrases can at times create confusing clusters. This is because we assume that phrases that frequently co-appear in the same documents are "similar", but this is not always true.

We have also noticed that the sizes and/or the number of "useful" clusters created by STC increases as more documents are retrieved. This means that for large document sets the users still have to scan many documents either because they scan many clusters or because the clusters they scan are large. As we are interested in making the system scale to substantially larger retrieved document sets, clusters should be presented hierarchically so users can navigate the results more efficiently.

Grouper II was designed to address these shortcomings. Grouper II has three basic components that are novel compared to Grouper I:

1. **Dynamic Index:** This new interface paradigm allows users to view non-merged phrase clusters. This view is essentially an inverted index of phrases to the retrieved document set. This view is useful in a higher portion of the queries (compared to the Grouper I interface), as users can easily ignore meaningless phrases and there are no mergers of phrase clusters that might be erroneous. We also believe this view is more intuitive to novice users. The dynamic index is generated by selecting a subset of the phrases identified by the suffix tree. We use a greedy algorithm to select a subset of high quality phrases, that also provide a high coverage (the percent of documents that include at least one of the phrases) of the results set.
2. **Multiple Views:** The system allows the user to view the documents in one of three interfaces: a dynamic index, clusters or a ranked list. Selecting the proper view allows the user to fit the interface to the informational task at hand.
3. **Interactive and Hierarchical Navigation:** Grouper II supports a hierarchical and interactive interface, similar to the Scatter/Gather interface [Cutting et. al., 92]. Users can select one or more index entries or clusters to focus on, thus defining a subset of the retrieved document set. Users then are able to recursively "zoom in" on this subset (viewing it via one of the three possible views). This feature allows users to navigate much larger document sets than the previous "flat" display of Grouper I.

Figure 1.2 presents the dynamic index interface, generated for the query "clinton". The dynamic index displays "interesting" phrases that were identified in the returned document set. The number of documents containing each phrase appears in parentheses besides it. The user can click on the checkboxes besides the phrases to mark the phrases that are of interest. Then, after selecting the next display mode, the user can "zoom in" on the subset of documents defined by the selected phrases. The "new query" input box is provided to allow the user to reformulate the query. A selected phrase is automatically added to the "new query" query box, as the phrases in the dynamic index can be very helpful in reformulating the query.

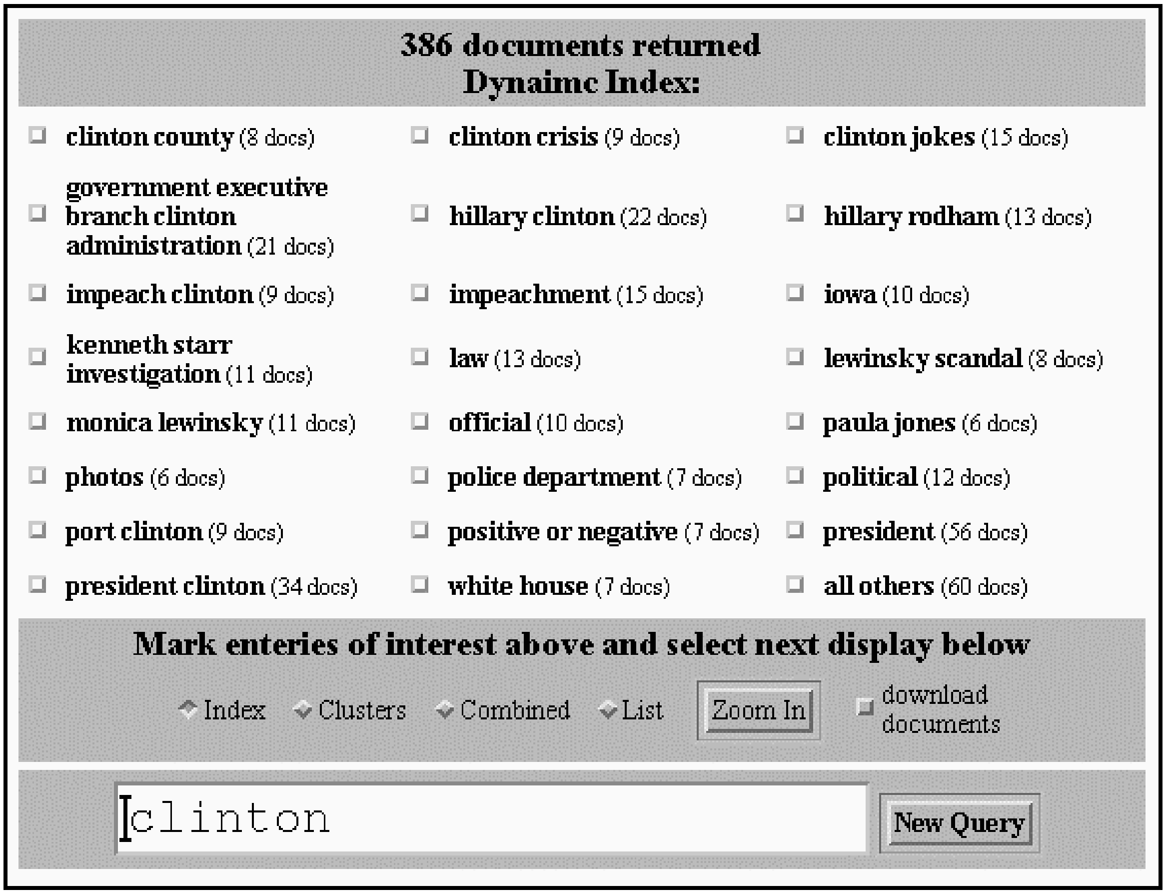


Figure 1.2: Grouper II's dynamic index interface

The dynamic index interface generated for the snippets returned by the query "clinton". The dynamic index displays "interesting" phrases that were identified in the returned document set. The "all others" entry designates all documents that do not include any of the phrases in the index. The user can click on the checkboxes besides the phrases to mark the phrases that are of interest. Then, after selecting the next display mode, the user can "zoom in" on the subset of documents defined by the selected phrases.

1.2.5 User Experiments

Is Grouper useful? How does it compare a ranked list interface (such a HuskySearch)? In what cases is it better? We addressed these questions in two ways.

First, we utilized the logs of the deployed Grouper I system as the basis of our evaluation. The logs record the behavior of a large number of Grouper users on queries of their choice. We compared these logs to the logs of the HuskySearch system (which is identical except for having a ranked list interface). Second, we performed a controlled user study using the Grouper II interface giving users specific tasks and specific retrieved document sets (thus evaluating purely their ability to browse the retrieved documents).

Analyzing the logs of the Grouper I system (including more than 3000 queries) we were able to find significant differences between the behaviors of Grouper I users and of HuskySearch users. First, Grouper user had fewer “dead-end sessions” (46% as compared to 53%). A dead-end session is a session in which the user does not follow even a single document (typically because she could not find anything of interest in the result set). Second, we observed that finding the first few interesting documents actually requires more “effort” in Grouper as compared to HuskySearch’s ranked-list presentation, but after the first two or three documents, finding additional interesting documents appears to require less effort in Grouper. Both time and a new metric called *click distance* were used to measure the effort of finding interesting documents. This possibly reflects the fact that the user must spend some time/effort to understand the clusters that Grouper presents, but after doing so the clusters are helpful in finding required information faster. It also confirms our observation that a clustering interface is not suitable for all search tasks.

By analyzing the logs of the Grouper II system we were able to show that compared to Grouper I it does a better job at placing together documents that seem interesting to the user. This is not surprising as Grouper II requires more user interaction than Grouper I. We have also demonstrated that finding multiple interesting document is faster using the Grouper II interface.

Our small-scale user experiment compared the dynamic index interface of Grouper II to a ranked list interface on a set of eight predetermined queries and the documents they retrieved. From this experiment we tentatively concluded that the dynamic index interface appeared to be most useful when the user required many documents relating to a certain information need. For such tasks Grouper II helped the user find more relevant documents (in a given amount of time) and to find them faster. We believe that the less frequent the relevant documents are within the document set, the more useful this interface becomes. When faced with the task of finding a single relevant document, the ranked list interface appeared to be more appropriate, though we suspect that when the relevant documents are very infrequent the dynamic index interface will actually be better.

1.3 Contributions

This thesis reports on the use of phrases to cluster Web search engine results. The scientific contributions of the thesis include:

1. **Using Post-Retrieval Document Clustering to Address the Low Precision of Web Searches**

   We have identified the advantages of document clustering as a tool to browse large document sets returned by low precision Web searches. We have defined the required characteristics of such a clustering system and its underlying clustering algorithm.

2. **Suffix Tree Clustering**

   We have presented a novel clustering algorithm – Suffix Tree Clustering – that is designed to address the requirements of Web search results clustering. Its key features include the use of phrases, and having a simple cluster definition that does not assume a specific model for the data.

3. **Evaluating the Use of Phrases in Clustering**

   We have shown that using phrases increases the clustering quality of various clustering algorithms, especially on the short and noisy Web documents. We have also compared the benefits of using different kinds of multiword features: n-grams, suffix tree phrases and frequent sets.

4. **Suggesting the Use of Frequent Sets in Information Retrieval**

   We introduced the possibility of using frequent sets as multiword features in IR. We demonstrated the advantages of using such features and presented modification to the standard generation algorithms to allow finding these sets faster.

5. **Implementing and Deploying a Document Clustering Interface to Web Searches:** We have implemented and deployed two generations of Grouper – a document-clustering interface to search results. The system combines three alternative interfaces: a clustering interface, a dynamic index, and a ranked list.

6. **Performing User Studies Comparing a Clustering Interface and a Ranked List:** We have carried out two user studies aimed at comparing a clustering interface and a ranked list interface. In the first study we analyzed the logs of the Grouper I system and compared them to the logs of HuskySearch. In the second study we performed a controlled user experiment comparing the dynamic index interface to a ranked list.

1.4 Organization

The remainder of this dissertation is organized as follows:

Chapter 2: Background

Chapter 2 reviews several topics related to this work. We present the problem of low precision searches on the Web and discuss why they are inevitable. We then review techniques Web search engines use to improve the relevancy of the document they retrieve, beyond the basic Information Retrieval technique of calculating a document's similarity to the query based on its text. We also describe what tools search engines currently offer users in an attempt to cope with low relevancy searches. Next, we review work on user interfaces to search results (various alternatives to the ranked list presentation), of which document clustering is but one technique. We then discuss document-clustering algorithms, illustrate the limitations of each algorithm and describe their use in Information Retrieval. Finally, we describe research on the use of phrases and multiword features in IR.

Chapter 3: Suffix Tree Clustering

Chapter 3 introduces the STC clustering algorithm. First, we detail the desired characteristics for a post-retrieval clustering algorithm and motivate the STC algorithm. We then describe the suffix tree data structure – its definition, characteristics and construction algorithms. Next, we describe the three stages of the STC algorithm: document parsing, identification of phrase clusters using a suffix tree and creating clusters by merging phrase cluster. We discuss STC's complexity as well as an incremental version of the algorithm. Finally, we detail some of the characteristics of the algorithm.

Chapter 4: Evaluation of Suffix Tree Clustering

In chapter 4 we empirically evaluate the STC algorithm. We start by describing the three document collections used in our experiments. We then describe experiments comparing STC with other commonly used clustering algorithm. We start by comparing the clustering quality of the different algorithms. Next, we investigate what contributes to the high quality of STC's clusters – we look at the contributions of using overlapping clusters and of using phrases. We also compare the performance of n-grams and suffix tree phrases. We then present Frequent Set Clustering (FSC) – a variant of STC that defines phrase clusters using frequent sets instead of suffix tree phrases – and investigate its clustering quality. Next, we compare STC to the other algorithms with regards to speed.

After evaluating the STC algorithm we turned our attention to investigate whether using phrases can improve the performance of vector-based clustering algorithm. Finally, we compare the performance of clustering algorithms on collections of search engine snippets and on their corresponding Web documents.

Chapter 5: Grouper I – First Implementation on the Web

Chapter 5 describes Grouper I – our first implementation of a document-clustering interface to search results. We describe the user interface of Grouper I, and explain key design decisions that make the system fast, and the clusters easy to understand. Next, we evaluate Grouper I in practice based on the logs gathered by this fully deployed system.

Chapter 6: Grouper II – The Next Generation

Chapter 6 introduces the Grouper II system – the second generation of our document clustering interface to a Web search results. We start by discussing some of the shortcomings of Grouper I that were identified during its deployment. Next, we describe the Grouper II system, explaining how it addresses these shortcomings, and present its user interface. Finally, we evaluate Grouper II based on its logs and describe a small-scale user study of system designed to investigate its usefulness.

Chapter 7: Conclusion and Future Work

Finally, Chapter 7 summarizes the results presented in this dissertation and gives directions for future work. These include discussing aspects relating to the possibility of commercializing Grouper, investigating the effect of snippet generation techniques on clustering quality, and incorporating relevancy information in the STC algorithm.

Chapter 2

BACKGROUND

2.1 Introduction

In this chapter we review several topics which are related to this work. We start by reviewing the techniques that Web search engines are using to improve the relevancy of the documents they retrieve, and what they currently offer users in an attempt to cope with low relevancy searches. We then review work on user interfaces to search results – alternatives to the ranked list presentation. Next, we cover document-clustering algorithms. Finally, we describe previous work regarding the use of phrases and multiword features in Information Retrieval.

2.2 Increasing Precision of Web Search Results

Document retrieval systems typically present search results ordered by their estimated relevance to the query, which is calculated using IR metrics that capture the similarity between the text of a document and the query. Such ranking schemes work well when users can formulate well-defined queries for their searches. However, users of Web search engines often formulate very short queries (70% are single word queries [Motro, 98]) that often retrieve a large set of documents. Based on such a condensed representation of the users' search interests, it is nearly impossible to locate within these large document sets the specific documents that are of interest to the user. Moreover, many webmasters now actively work to influence rankings. These problems are exacerbated when the users are unfamiliar with the topic they are querying about, when they are novices at performing IR searches, and when the database contains a large number of documents. All these conditions commonly exist for Web search engine users.

To address these problems, recent research has focused on ranking retrieved results based on information other than the text appearing in the documents.

The Google search engine makes use of the hyperlink graph structure of the Web in order to determine an a priori page relevance, which is later combined with its relevance to the query [Brin and Page, 1998]. Google uses a ranking algorithm called PageRank [Page et al., 98] that iteratively ranks a page's importance based on the number of pages that point to it and the importance of those pages. The PageRank PR(p) of a page p, which has pages l_1 through l_n pointing to it is:

$$PR(p) = (1\text{-}d) + d(\ PR(l_1)/C(l_1) + \ldots + PR(l_n)/C(l_n)\ )$$

with d a damping factor between 0 and 1, and $C(l_i)$ defined as the number of links going out of page l_i. The PageRanks form a probability distribution over Web pages, so the sum of all pages' PageRanks will be one. The PageRank of all pages can be calculated using an iterative algorithm, and corresponds to the principal eigenvector of the normalized link matrix of the indexed pages. Google also makes extensive use of the text within hyperlinks. This text is associated with the page the link points at, and it makes it possible for Google to find matching pages even when these pages do not contain text that successfully characterizes their contents (e.g., pages containing only pictures).

Laser (lazer.cs.cmu.edu) [Boyan et al., 96] also uses the hyperlink structure but in a different way. By analogy to reinforcement learning, they view the similarity of a page to the query as the "immediate reward", and propagate these rewards back through the hyperlink graph, discounting them at each step.

Clever (http://www.almaden.ibm.com/cs/k53/clever.html) uses the HITS (Hypertext-Induced Topic Search) algorithm to locate two types of useful pages – *authorities*, which contain a lot of information about a topic, and *hubs*, which contain a large number of links to pages on the topic [Kleinberg, 98]. The underlying principle is the following: good hub pages point to many good authority pages, and good authority pages are pointed to by many good hub pages. A Clever search begins with the collection of several hundred retrieved pages (using standard scoring techniques). Traversing links from this initial set adds additional pages to this collection. Hub and authority scores are then calculated for all the pages and these scores are then used to rank the pages. Clever also considers the text within and near each link – if the actual search terms appear, then that link transmits more weight to the page it points to.

Hotbot combines standard search technology with a popularity score supplied by Direct Hit (www.directhit.com). Direct Hit tracks which search result links users click and how long users stay at each site, *i.e.* the more popular pages are ranked higher. The system is ideally suited for general queries of one or two words, which are very common on the Web.

Infoseek uses its "ESP" ranking system, which gives more weight to pages which come from sites that are also listed in Infoseek's human-compiled directory, or which have some degree of link popularity, or both. GoTo uses a (highly debated) pay-for-placement model. In a recent survey of major search engines GoTo ranked number one for users reporting they found what they were looking for "every time" (independent online consumer survey conducted by NPD Online Research at the beginning of 1999). AltaVista added a similar feature for several months, but backed away because of public outcry.

2.3 Helping the User in Low Precision Searches

Many Web queries result in a very large number of returned documents. Typically, the vast majority of these are of no interest to the user and therefore such searches can be termed *low precision searches*. Most Web search engines provide tools aimed at helping users “cope” with (and make use of) these large document sets.

Many Search engines allow users to sort the results by site (e.g., Excite, Infoseek, Hotbot and Lycos), or by date (e.g., InfoSeek and NorthernLight). Some search engines allow users to narrow down the set of matches already generated (the “Search Within” feature of Infoseek and Lycos).

Another approach is to provide a “Similar Searches” feature, which displays popular queries that are related to the user's original search (*e.g.*, Infoseek, Altavista, Hotbot, Excite). The similar searches are displayed as links, so that clicking on a similar search will rerun the query using those terms. Excite also displays words that appear related to the query (words that frequently co-occur with the query terms). The user can choose to add these to the search box by clicking a checkbox next to each word. Several search engines offer the ability to seek out other pages that are similar to a particular page that the user likes (*e.g.*, Excite, Infoseek and Lycos). By choosing this option, a new search is performed with the "originating page" serving as the query. Search engines containing directories (*e.g.*, Infoseek and Yahoo!) typically match the users' queries with topics within their directories. Users can abandon their search results altogether and look for relevant documents within these topics.

NorthernLight provides "Custom Search Folders", in which the retrieved documents are organized. These browsable folders are organized hierarchically and can be based on type (*e.g.*, press releases, product reviews, etc.), source (sites, domains), language or subject. In order to create subject folders, NorthernLight assigns a set of topic tags (potential folders) to each page as it indexes the page. This is done by classifying the page against one or more values from a 20,000 term subject hierarchy manually developed by librarians. The subject hierarchy was built from sources ranging from the Library of Congress subject headings to drug store aisle signs [Krellenstein, 98]. When a query is processed, the system searches for the dominant topic tags among the matching pages and selects from these a subset of tags that are presented as folders. Folders are selected based on the number of documents in each folder, their relevance to the query and their variety. NorthernLight does not disclose the precise classification algorithm or the folder selection algorithm. Electric Library (www.elibrary.com) uses "Recurring Themes" in a similar manner. Themes are of three categories: places, persons and subjects. The subject-themes appear to be selected from a restricted, predefined list.

It is interesting to note the difference between Grouper and NorthernLight. First, Grouper receives hits from different engines, and only processes the top hits from each search engine. Thus, its clusters are not affected by many low relevance documents, as is the case for NorthernLight's folders. Second, Grouper performs post-retrieval clustering, thus the reaction time is slightly longer, but the clusters are tailored to the retrieved set and are typically of higher quality (see section 2.5.3). Third, NorthernLight relies on a static, manually constructed hierarchy of topics, which must be constantly maintained and updated. Grouper relies on the automatic detection of phrases and is therefore content driven. It would work just as well in German whereas NorthernLight would not. Finally, as Grouper can run on a client machine, it is spider and search engine independent and does not require additional resources from the server. This allows it to be easily adapted to any search engine and to be used in meta search engines.

We shall now present several examples to illustrate the difference between the groups created by NorthernLight and by Grouper. Figure 2.1 presents the groups (phrase clusters in Grouper; subject folder in NorthernLight – other types of folders, such as source folders, were removed as they are not relevant to this comparison) displayed to the user in the two systems in response to the query "Lewinsky" (on 9/10/99). NorthernLight found 7 subject folders. The folder "Starr, Ringo" is an example of a topic that has no relation to the query but was still "triggered" (presumably by the word "Starr"). The other folders (*e.g.*, "White House" and "Starr report") seem coherent and related to the query. However we believe the phrase clusters that were found by Grouper to be more interesting. Clusters such as "fan club" and "immunity from prosecution" seem very appropriate for a person looking for information about Lewinsky. On the other hand, the "Starr report" folder is somewhat distinct from the Lewinsky topic. If the user were interested in the Starr report, would the query contain only the word "Lewinsky"?

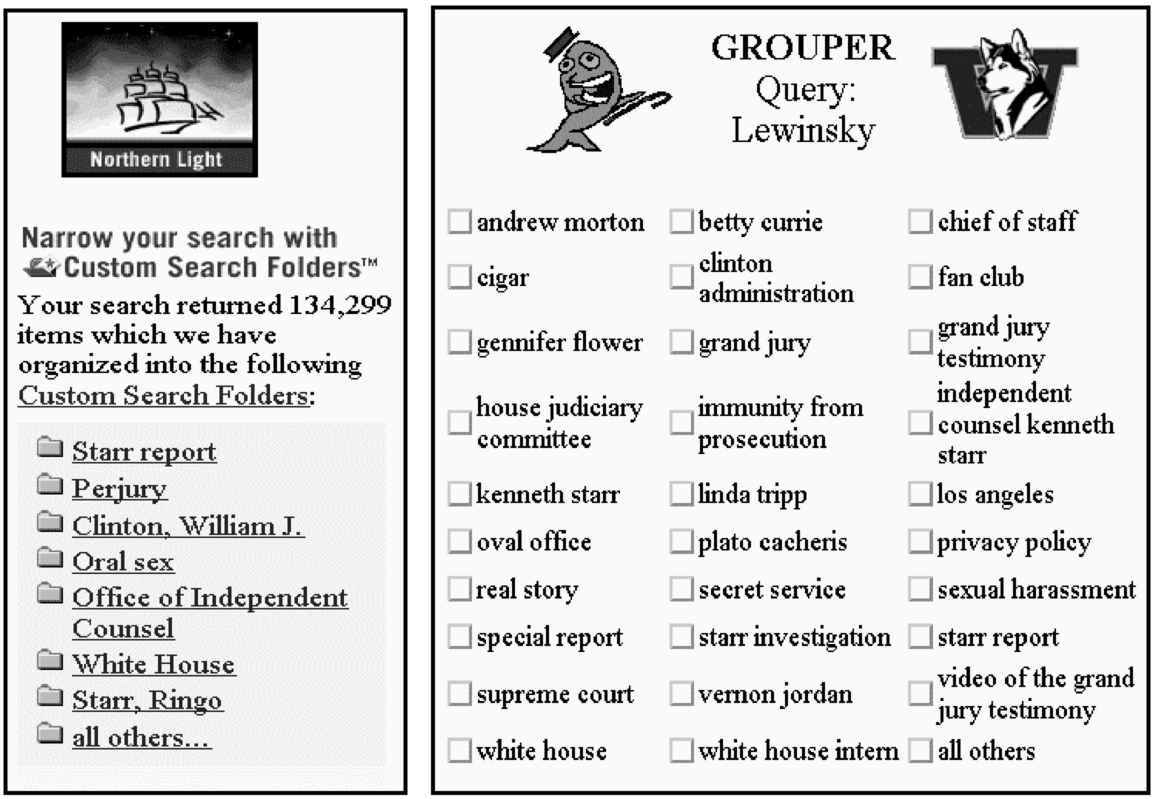


Figure 2.1: NorthernLight vs. Grouper

Comparing the output of NorthernLight and Grouper in response to the query "Lewinsky" (on 9/10/99). The Figure presents the groups (phrase clusters in Grouper; subject folder in NorthernLight – other types of folders, such as source folders, were removed as they are not relevant to this comparison) displayed by the two systems.

An example that NorthernLight seems to perform well on is the query "salsa". The folders displayed by NorthernLight include "Dance", "Gourmet food", "Cha-cha", "Music products and services", "Restaurants" and "Latinos". These are all well-defined categories that can help the user focus her search. On the other hand when given the query "Turkey Earthquake" (this was approximately three weeks after the 8/17/99 earthquake in Turkey that killed 14,000 people), NorthernLight was able to find only two subject folders – "Earthquakes" and "Petroleum industry". Given the same query, Grouper displayed phrase clusters such as "American red cross international response fund", “credit card donation”, “death toll”, “Kandilli observatory and earthquake research” (the only earthquake research institute in Turkey), “relief effort”, “Richter scale”, “Turkish prime minister” and “United Nation”. The reason for the poor performance of NorthernLight on this query was that few of its predefined topic tags were appropriate for the topic of this search.

2.4 User Interfaces to Search Results

Visualization of search results has been investigated as a tool for presenting retrieved documents to the user in ways that can scale to large document sets and provide more information to the user than the ranked list interface. Various visualization techniques were designed to help users get a better comprehension of the returned document set, identify interesting documents faster, and reformulate the query more accurately.

Document visualization techniques fall into two broad categories: visualization of document attributes, and visualization of interdocument similarities.

The visualization of document attributes is designed to display additional information about the retrieved documents. This will often have the secondary effect of grouping documents that share similar attributes. We can further classify these interfaces depending on the additional information they present to the user:

1. **Query Terms' Distribution:** In best-match document retrieval systems it is often difficult to understand why a document was retrieved or how its relevance score was calculated. Some interfaces show how the retrieved documents relate to each of the terms used in the query, as a means of addressing this problem [Veerasamy & Belkin, 96; Veerasamy & Heikes, 97]. Other systems also show how query terms are distributed within long documents [Hearst, 95].
2. **Document Attributes:** The interfaces in this category show how the retrieved documents relate to predefined document attributes such as size, date, source, author or popularity [Card et al., 91; Nowell et al., 96]. Some systems have categories pre-

assigned to each document, and the retrieved documents are displayed with regards to these labels [Hearst, 94; Kim & Korfhage, 94; Rose & Belew, 91; Hearst & Karadi, 97]. Some systems also use user-specified attributes, such as previous queries, user profiles or selected reference documents. [Spoerri, 93; Korfhage, 91; Nuchprayoon & Korfhage, 97; Olsen et al., 93; Hemmje et al., 94].

The second category of visualization techniques – the visualization of inter-document similarities – reduces the dimensionality of the document-space so users will be able to visually perceive the relationships between the documents. This can help users get an overview of the collection or find relevant documents faster, as once an interesting document has been found it is easy to find others similar to it by investigating its neighbors in the visualized space. We can further classify interfaces of this category by the methods they use to achieve this dimensional reduction:

1. **Document Networks:** These interfaces display the document set as a network, with nodes representing documents, and links representing a degree of similarity [Thompson & Croft, 89; Fowler et al., 91; Fowler et al., 95; Mukherjea et al., 95].
2. **Spring Embeddings:** These interfaces use algorithms that model the document set by a system of particles that exert repelling and attracting forces on each other based on the similarity of their corresponding documents [Chalmers & Chiton, 92; Chalmers, 96]. The usefulness of such an interface was demonstrated by the AspInquery Plus system [Swan & Allan, 98] which was part of the TREC-4 aspect oriented retrieval task, wherein users were required to find, in a given period of time, as many different aspects as possible of a specified information goal. Users of the system were able to better decide which documents to investigate based on their proximity to previously investigated documents.
3. **Clustering:** Clustering algorithms attempt to identify groups of documents that are similar to each other more than they are similar to the rest of the collection. These methods will be covered in more detail in the next section.

4. **Self-Organizing Maps (SOMs):** These algorithms map the document collection onto a 2-D surface so that similar documents are mapped close to each other. The Kohonen Self-Organizing Map [Kohonen, 90] is a general neural network algorithm for positioning high-dimensional data so that alike inputs are in general mapped close to each other. Lin was the first to apply these maps to document sets [Lin, 91; Lin, 93; Lin, 97]. The WEBSOM project [Kohonen, 97] showed that SOMs can be scaled to display over 1,000,000 documents. Hierarchical SOMs have been used to speed up the learning phase of the algorithm [Merkl, 97].

2.5 Document Clustering

In this section we review document clustering. We start by reviewing clustering algorithms, their complexity and their shortcomings. We then illustrate the limitations of each algorithm by providing graphical examples showing why they might fail (i.e., create clusters that are "intuitively" wrong). Finally, we discuss document-clustering research in Information Retrieval.

2.5.1 Clustering Algorithms

Numerous document clustering algorithms appear in the literature (see [Willet, 88] for a review). These can be classified into two groups – those producing hierarchical clusters and those producing a flat partition (note that partition algorithms can be applied recursively to create a hierarchy of clusters). Most clustering algorithms rely on an external similarity measure between documents. This is typically calculated by representing each document as a weighted attribute vector, with each word in the entire document collection being an attribute in this vector (the *vector-space model* [Salton, 89]). The similarity of two documents is often taken as a normalized function of the dot product of their attribute vectors (*e.g.*, the cosine similarity measure).

Hierarchical Agglomerative Clustering (HAC) algorithms have been widely applied.

These algorithms start with each document in a cluster of its own, iterate by merging the two most similar clusters, and terminate when some halting criterion is reached. Algorithms differ based on their definition of cluster similarity. The *single-link* and *complete-link* algorithms define the similarity of two clusters as the minimal and maximal similarity (respectively) of any pair of items, one from each cluster. The *group-average* algorithm defines it as the average similarity of all pairs of items. These algorithms are typically slow when applied to large document collections; single-link and group-average can be implemented in $O(n^2)$ time (where n is the number of item), whereas complete-link requires $O(n^3)$ time [Voorhees, 86]. These algorithms are too slow to meet the speed requirement for online interaction when clustering several hundred items. In terms of quality, on the other hand, complete-link has been shown to perform well in comparative studies of document retrieval [Voorhees, 85], as it tend to produce "tight" clusters – clusters in which all the documents are similar to one another. Single-link, and to a lesser degree group-average, exhibit a tendency of creating *elongated* clusters, which have the undesirable property that two documents can be in the same cluster even though the similarity between them is small. In "noisy" domains (such as the Web), algorithms that produce elongated clusters often result in one or two large clusters, and many extremely small ones [Griffiths et al., 86]. This is typically not what users expect. Several halting criteria for HAC algorithms have been suggested [Milligan and Cooper, 85], but in practice predetermined constants (*e.g.*, halt when 5 clusters remain) are typically used. HAC algorithms are very sensitive to the halting criterion – when the algorithm mistakenly merges multiple "good" clusters, the resulting cluster could be meaningless to the user. In the Web domain, where the results of queries could be extremely varied (in the number, length, type and relevance of the documents), this sensitivity to the halting criterion often causes poor results.

Another class of algorithms is the partition algorithms. Most of these are also known as *model-based* algorithms as they have a priori assumptions as to the model describing the data. These algorithms search for the most probable model parameters, given the data and the a priori assumptions regarding the model. Of these, k-means [Rocchio, 66] is the most basic. It assumes a given number of clusters that are spherical (with respect to the document similarity metric used) and of equal size. However, there is no reason to assume that “good” document clusters observe this model. The k-means algorithm uses the EM (Estimation-Maximization) approach to find the most probable model parameters. This process starts by randomly selecting initial cluster centroids (the “centers of mass” or the representative vectors of the clusters). Then iteratively, it assigns documents to clusters and recalculates the cluster centroids. K-means is a fast, linear time algorithm (actually $O(nkT)$ time complexity where k is the number of desired clusters and T is the number of iterations), but it has an undesired element of randomness.

Unsupervised naïve-bayesian classification [Cheeseman et al., 88] is another model-based algorithm. It is more sophisticated than k-means as it not only searches for the most probable model parameters given the data and prior expectations, but also for the most probable model. For example, bayesian clustering can produce clusters of different sizes, with a non-unity covariance metric, and with variable distributions. This approach has produced good results in several domains [Cheeseman and Stutz, 96], but hasn’t been widely applied to document clustering. Bayesian clustering also uses the EM method to solve the optimization problem, and is linear in the number of documents (though with a much larger constant than the k-means algorithm, as the number of variables in the search space is much greater).

The *single-pass* algorithm [Hill, 68] attempt, like k-means, to find spherical clusters of equal size. It is an incremental algorithm that uses a greedy search, assigning each document to a cluster only once. The first document processed is used to start the first cluster. Every additional document is compared to all existing clusters, and its most similar cluster is found. If its similarity to this cluster is above a predefined threshold the document will be added to that cluster, otherwise it will be used to create a new cluster. Single-pass is a linear time algorithm. It suffers from being order dependent, from being extremely sensitive to the predefined threshold, and from having a tendency to produce large clusters [Rasmussen, 92]. It is, however, the most popular incremental clustering algorithm (as can be seen from its popularity in the topic detection domain [TDT, 97]).

Buckshot and Fractionation [Cutting et al., 92] are also linear time algorithms. Fractionation is an approximation of HAC, where the search for the two closest clusters is performed locally (among a subset of the clusters), not globally. Buckshot is a k-means algorithm where the initial cluster centroids are not chosen at random but created by applying HAC clustering to a sample of the documents of the collection.

2.5.2 Examples of Clustering Algorithms Failing

In the previous section we introduced the major clustering algorithms, each with its own limitations. In this section we shall illustrate the limitations of each algorithm by providing graphical examples showing where they might fail (*i.e.*, create clusters that are "intuitively" awkward). The examples shown are not meant to convey the patterns found in retrieved document sets – they are presented to help illustrate the explicit and implicit assumptions of different clustering algorithms.

Most of the algorithms presented in the previous section create spherical, equal-sized clusters. These include the group-average and the complete-link HAC algorithms and the k-means and single-pass partition algorithms. When faced with data that does not fit this model, these algorithms often perform poorly. Figure 2.2 illustrates this issue. In this figure points are scattered on a two-dimensional plane, with their geometric distance representing their similarity. If group-average, complete-link, k-means or single-pass were run to find three clusters in the data, the results would be as shown in the Figure 2.2 – the large cluster would be split into two, and the two smaller clusters would be clustered together (along with some points from the big cluster).

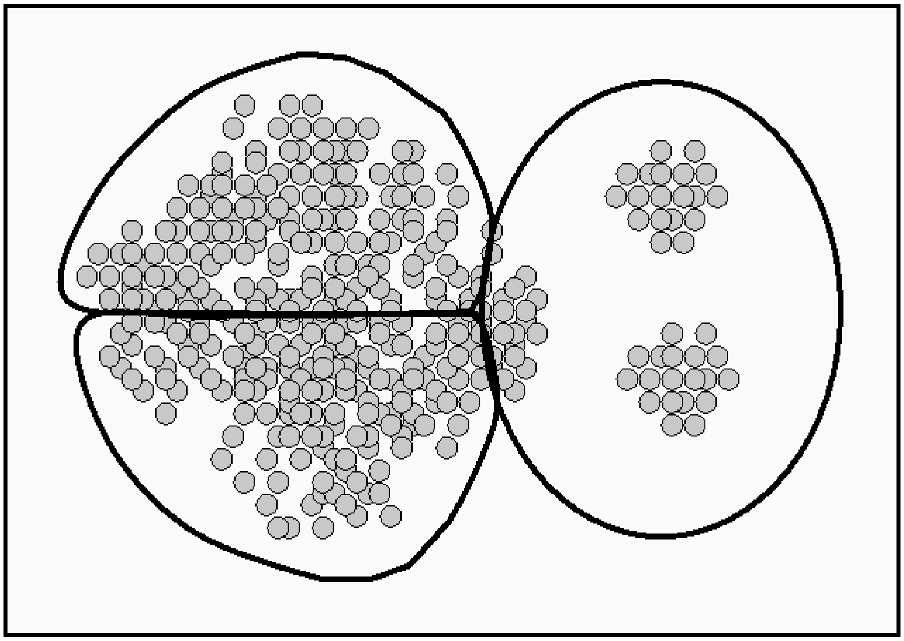


Figure 2.2: Clustering data from unequal-sized clusters

The clusters found by group-average, complete-link, k-means and single-pass algorithms when searching for three clusters. These algorithms attempt to create spherical, equal-sized clusters, but the data in the figure does not fit this model.

We have previously mentioned that HAC algorithms are very sensitive to outliers. If we add some noise to the data in Figure 2.2 the resulting clusters might be quite different. Figure 2.3 presents the same data presented in Figure 2.2, with the addition of six new outlier points. The resulting clusters for all three HAC algorithms are presented in Figure 2.3. The HAC algorithms place the six new points in two clusters, and mark all the previously existing points as a single cluster.

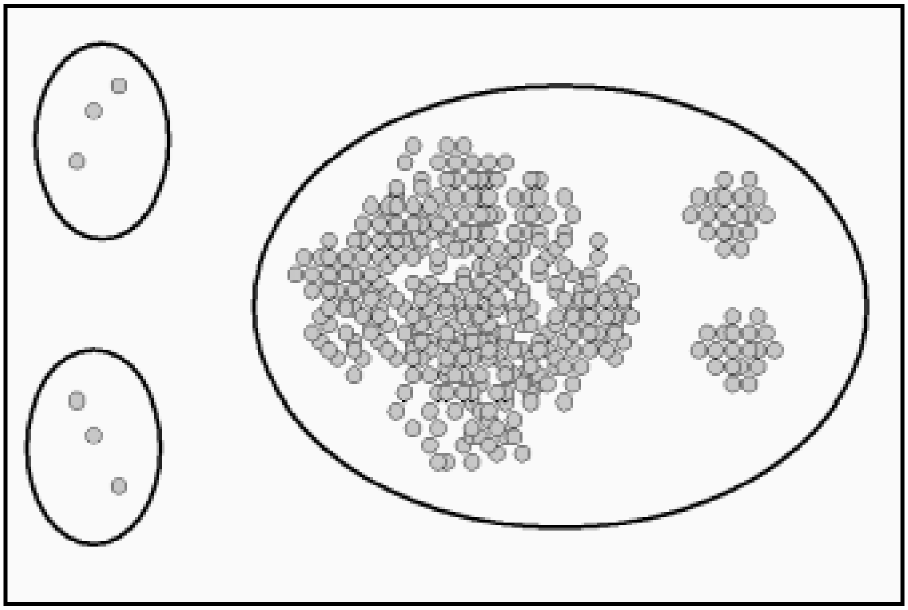


Figure 2.3: Clustering data with outliers

The clusters found by the single-link, group-average and complete-link HAC algorithms when searching for three clusters. These algorithms are very sensitive to outliers. The data in the figure is the same data presented in Figure 2.2 with the addition of six new outlier points. The HAC algorithms place these outliers in two clusters and all the rest of the data in a single cluster.

Figure 2.4 illustrates a case in which the points are not positioned based on a spherical model. All the model-based algorithms would fail in this case, producing meaningless results. Even the more robust bayesian-clustering algorithm would probably blunder, fitting the data with the most appropriate model it might find. The circles in Figure 2.4 illustrate a possible model that a bayesian clustering algorithm might find the most probable given the data.

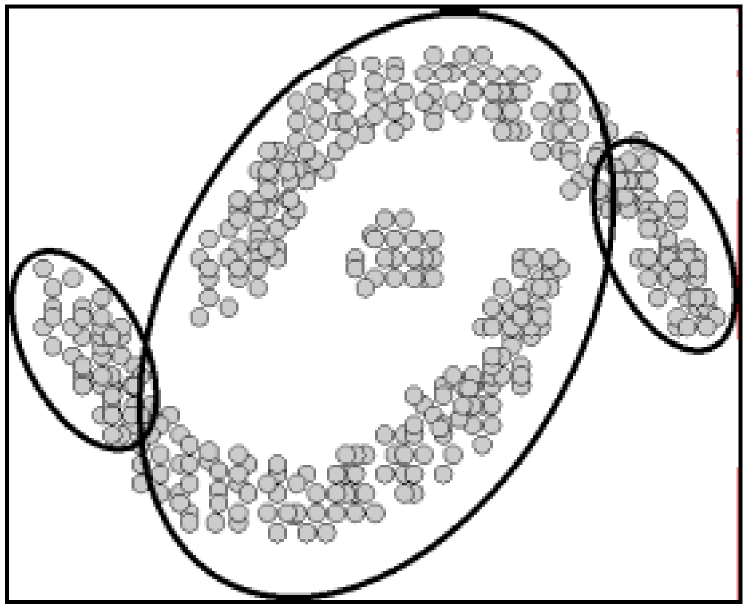


Figure 2.4: Bayesian clustering of data from non-spherical clusters

The figure illustrates a case in which the points are not distributed based on a spherical model. All the model-based algorithms would fail in this case, producing meaningless results. Even the more robust bayesian-clustering algorithm would probably blunder (this is of course a subjective assessment which relies on the expectations of the user). The circles illustrate a possible model that a bayesian clustering algorithm might find as being the most probable given the data.

Figure 2.5 is an example of the chaining effect of the single-link algorithm. Given the data in the figure and asked to find two clusters, single link would locate one small cluster (the one circled with a black line) and one very large cluster (all the rest of the data points). This might actually be the desired answer (*e.g.*, when points represent trees and we are conducting research as to the variability of the gene pool in that area regarding a species that never traverses open space), but it is not necessarily suitable for most domains.

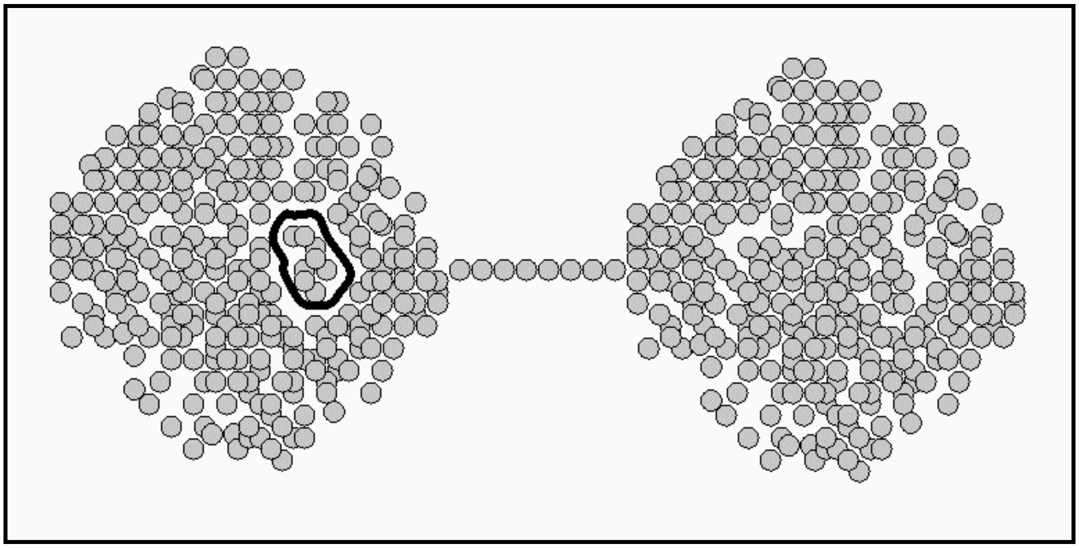


Figure 2.5: The chaining effect of the single-link algorithm

Given the data in the figure and asked to find two clusters, single link would locate one small cluster (the points circled with a black line) and one very large cluster (the rest of the points). This might not be the desired result in the document-clustering domain.

CURE is a recently introduced HAC algorithm that was designed to address some of the problems illustrated above [Guha et al., 98]. It is a hybrid between the single-link and the group-average algorithms, thus it is better able (compared to group-average) to identify clusters of various shapes and sizes, with a limited chaining effect (compared to single-link). It also applies heuristics for outlier elimination. CURE represents each cluster with a set of representative points, generated by choosing a set of *well scattered* points from the cluster, and then shrinking these towards the cluster's centroid by a certain factor. The distance between two clusters is defined as the distance between the two closest representative points, one from each cluster. This algorithm has currently been tested only on synthetic data (points on a plane – not documents) designed to show its strengths.

2.5.3 Document Clustering for Information Retrieval

Document clustering has initially been investigated in Information Retrieval mainly as a means of improving the performance of search engines by pre-clustering the entire corpus [Jardine and van Rijsbergen, 71; Salton, 71; Croft, 78; Griffiths et al., 86]. The *cluster hypothesis* [van Rijsbergen, 79] stated that similar documents will tend to be relevant to the same queries, thus the automatic detection of clusters of similar documents can improve recall by effectively broadening a search request. However, this approach has not been shown to be superior to standard near-neighbor searches.

The Scatter/Gather system [Cutting et al., 92] introduced clustering as a document browsing method. It used a linear time clustering algorithm (Buckshot) to cluster a corpus of documents and presented these clusters to the user. The user selected one or more clusters for further investigation, and the documents in this subcollection were then reclustered in an iterative and hierarchical manner.

Clustering has recently been used for browsing search results [Allen et al., 93; Leouski and Croft, 96]. The Scatter/Gather system has also been applied in this context [Hearst & Pedersen, 96]. It was the first to use fast, linear time algorithms and thus was able to cluster sufficiently fast a reasonable number of search results (1000 abstracts in less than one minute).

A second approach has been to group retrieval results using a pre-computed hierarchy of clusters of the entire corpus [Cutting et al., 93]. As the pre-computed clusters are not always "suitable" for the retrieval results, additional processing might be needed to improve the quality of the clusters [Silverman & Pedersen, 97]. This is done by first finding a set of clusters in the existing cluster hierarchy that represent a reasonable embedding of the retrieved documents. Large clusters are expanded and replaced by their children, resulting in a predefined constant number of clusters. A clustering algorithm is then applied to these predefined clusters (merging similar clusters together). Finally, documents that were not part of the retrieval results are removed from the generated clusters. This method resulted in a significant speedup compared to an algorithm that simply clusters the retrieved document set (15 seconds compared to 32 seconds for 1000 document) without much loss in quality.

Since the Web is a dynamic environment, static, pre-computed clusters would have to be constantly updated, and most clustering algorithms can not do so incrementally. This would require a huge amount of resources. A pre-computed clustering of a sizeable part the Web (30M Web documents in over 10 CPU-days) was shown to be feasible [Broder et al., 97], but the approach used (a non-hierarchical single-link algorithm with a predetermined similarity threshold) was designed for near-duplicate document identification and is not suitable for the grouping of retrieval results.

Figure 2.6 illustrates why post-retrieval clustering is often superior to the selection of pre-computed clusters. Assume the small circles represent documents (the geometrical distance representing document similarity), and the whole corpus is pre-clustered hierarchically into four clusters. Assume a given query retrieved five documents, the ones colored in gray (marked 1-5), and we wish to cluster them into two clusters. We define good clusters as clusters with maximal inter-cluster similarity. An algorithm that must select from the preexisting cluster structure will create clusters corresponding to the two sides of the tree: one containing documents 1 and 2, and another containing documents 3, 4 and 5. A post-retrieval clustering algorithm, on the other hand, should generate a cluster containing documents 1, 2 and 3 and another containing document 4 and 5. The latter clusters are superior, as they minimize inter-cluster document distances (thus maximizing inter-cluster document similarity). Post-retrieval clustering can thus focus on features that are dominant in the result set, while pre-retrieval clusters might be influenced by features dominant in the corpus as a whole, but not frequent in the result set. NorthernLight can be viewed as using a pre-retrieval clustering technique as folders are selected from pre-assigned topic tags. They alleviate this potential problem by assigning multiple topic tags to each document and then selecting the most “fitting” topics based on the retrieved set.

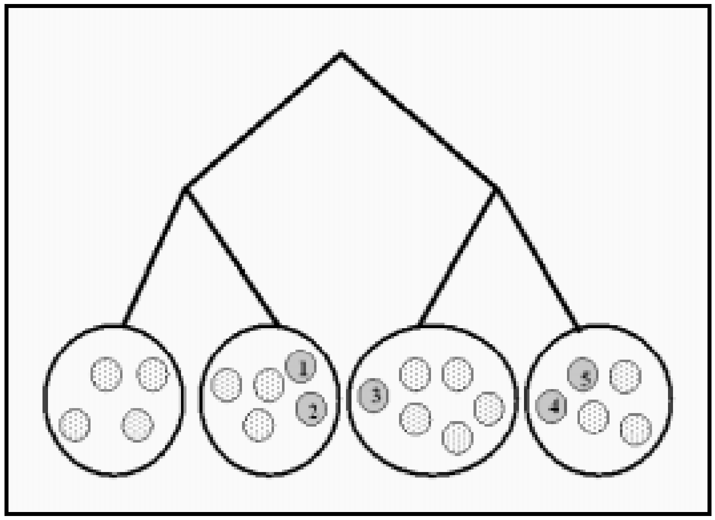


Figure 2.6: Post-retrieval vs. pre-retrieval clustering

An illustration why post-retrieval clustering is often superior to the selection of pre-computed clusters. The small circles represent documents, and the corpus is pre-clustered hierarchically to into four clusters. The gray circles represent document retrieved by a query (numbered 1-5). A pre-retrieval clustering algorithm will place document number 3 together with documents 4 and 5 (based on the pre-computed cluster hierarchy). A post-retrieval clustering algorithm, on the other hand, will place it with documents 1 and 2. The latter clusters are superior, as they maximize inter-cluster document similarity (which is our measure of clustering quality in this example).

2.6 Phrases and Multiword Features in Information Retrieval

The STC algorithm identifies phrases that are common in the document set, and uses these phrases as the basis for creating clusters. In contrast to STC, all the above mentioned algorithms treat a document as a set of words and not as an ordered sequence of words, thus losing valuable information. In this section we shall review research regarding the use of phrases and multiword features in IR.

Phrases have long been used to supplement word-based indexing in IR systems. Syntactic phrases are generated using syntactic parsing to find words in a particular syntactic relationship. These techniques break down into two major categories: template-based and parser-based. Template-based techniques attempt to match adjacent words against a library of templates such as <JJ-NN NN> (adjective noun) and <NN PP NN> (noun preposition noun) [Dillon and Grey, 83]. Parser-based systems attempt to analyze entire sentences. Recently, the use of syntactic phrases has shown a consistent and significant improvement in retrieval performance (typically improving precision without hurting recall) [Zhai et al., 97; Strzalkowski et al., 97].

Statistical phrases are generated by simple statistical approaches such as contiguous non-stopped words [Salton et al, 75], contiguous pairs of words that co-occur frequently [Fagan, 87] or “significant” n-grams (based on mutual information metrics) [Brown et al., 90]. Multiword features consisting of pairs of words within a certain distance of each other have also been used (and are often regarded as statistical phrases) [Hull et al., 97]. Statistical phrases have been shown to improve performance, but not as much as syntactical phrases [Hull et al., 97].

Phrases have also been extensively used in classification, though with mixed results. Lewis concluded that phrasal features provide no advantage over conventional word features [Lewis, 92]. More recently, using template-based syntactic phrases to classify Web pages improved precision at the expense of recall [Furnkranz et al., 98]. Using n-grams has shown minor improvements in both precision and recall in two studies [Furnkranz, 98; Mladenic and Grobelnik, 98]. It is interesting to note that the improvement was achieved by adding 2-grams (a slight improvement was also achieved by adding 3-grams). Longer n-grams actually decreased performance. In the TREC routing track using word-pair features (within a certain number of words) also showed an increase of performance of about 10% [Allan et al., 96].

The use of phrases or multiword features in document clustering is less common – the author knows of only one such research endeavor. Pairs of words that co-appear within a sliding window of five words (termed *lexical affinities*) were used as the attributes of the documents' vector representations (instead of single words) [Maarek and Wecker, 94]. A standard HAC algorithm was then applied, producing better results than when using vector representations with single words as attributes.

On the Web, AltaVista and Google both offer semi-automatic phrase searching. This can be helpful, because a phrase search can often yield more relevant results than an ordinary search. AltaVista tries to extract phrases from the terms the user enters using a large dictionary of common phrases. When a search is entered AltaVista checks whether it contains any phrases that are in the dictionary. If so, it will perform a phrase search automatically. If not, it will default to a normal search. Google applies a similar strategy, trying to find phrases in the entered query.

Chapter 3

SUFFIX TREE CLUSTERING

3.1 Introduction

In this chapter, we introduce *Suffix Tree Clustering* (STC) – a novel clustering algorithm designed to meet the requirements of post-retrieval clustering of Web search results. STC is unique in treating a document as a string, not simply a set of words, thus making use of proximity information between words. STC relies on a *suffix tree* to efficiently identify sets of documents that share common phrases and uses this information to create clusters and to succinctly summarize their contents for users.

First, we describe the desired characteristics for a post-retrieval clustering algorithm and motivate the STC algorithm. We then describe the suffix tree data structure: its definition, characteristics and construction algorithms. Next, we describe the STC algorithm and its complexity. Finally, we detail some of the characteristics of STC.

3.2 Desiderata and Motivation for the STC Algorithm

Many document clustering algorithms rely on the offline, pre-retrieval clustering of the entire document collection [Cutting et al., 93; Silverstein and Pedersen, 97], but the Web is too large and fluid to allow offline clustering.

The NorthernLight search engine tags all documents, at indexing time, with topic labels from a fixed set, and then displays query results with respect to these topic labels. Such an approach, though fast, suffers from the limitations of pre-retrieval clustering mentioned in the previous chapter: the clusters are not determined by "local" patterns in the results set (but rather by "global" patterns in the whole document collection that might not be significant in the results set). Another drawback to this approach is that the predetermined topic labels might not be appropriate for the users' search goals (in post-retrieval clustering, on the other hand, the clusters are determined based on features identified in the retrieved document set).

Our approach is to cluster the much smaller set of retrieved documents. Because search engines service millions of queries per day, free of charge, the CPU cycles and memory dedicated to each individual query are severely curtailed. Thus, we have adopted a model in which the clustering is performed on a separate machine, which receives search engine results as input, creates clusters and presents them to the user. This separate machine can be an intermediate server (such as our Grouper systems), or it can be the user's client. Another advantage of such a model is that the clustering system does not have to be integrated with the search engine. In chapter 7, we will discuss what advantages can be gained from altering this model and allowing more complex integration with the search engine.

We have identified three key requirements for a post-retrieval document clustering system:

1. **Relevance:** The system ought to produce clusters that group similar documents together (or stated in an IR fashion: documents relevant to the user's query should be grouped separately from the irrelevant ones).
2. **Browsable Summaries:** The user needs to determine at a glance whether a cluster's contents are of interest. We do not want to replace sifting through ranked lists with sifting through clusters. Therefore, the system has to provide concise and accurate descriptions of the clusters.
3. **Speed**: As this is an online system, there should be little or no apparent delay to the user. A very patient user might sift through one hundred documents in a ranked list presentation. We want clustering to allow the user to browse through at least an order of magnitude more documents. Therefore the clustering system ought to be able to cluster up to one thousand document in a few seconds. For the impatient user, each second counts. This entails that asides from having a fast and scalable

clustering algorithm (*e.g.*, it can not be quadratic in the number of documents), the system should also be *Snippet tolerant*: it ought to produce high quality clusters even when it only has access to the snippets returned by the search engines, as most users are unwilling to wait while the system downloads the original documents off the Web. Another important characteristic of the system should be *Incrementality*: to save time, it should start to process each document as soon as it is received over the Web.

These basic requirements suggest several additional important features for the clustering algorithm:

1. **Overlapping Clusters:** To increase the ability of the clustering algorithm to place similar documents in the same cluster, the algorithm should allow overlapping clusters. It is important to avoid confining each document to only one cluster since documents often have multiple topics, and thus might be similar to more than one group of documents [Hearst, 98]. This issue also relates to the fact that there is no symmetry between the two classes of errors a clustering algorithm produces. Users can usually easily ignore a document that is placed erroneously in an unrelated cluster, but documents that are missing from their appropriate clusters will often be totally irrecoverable to the user. Allowing overlapping clusters can be viewed as preferring errors of the first type to errors of the second type.
2. **Phrases:** Most clustering algorithms treat a document as a set of words ("bag of words"), thus ignoring important information regarding word proximity. A clustering algorithm that looks at phrases in the documents can utilize this additional information. Using phrases has two important benefits: First, it can improve the quality of the clusters, by utilizing more information present in the text of the documents. Second, phrases are useful in constructing concise and accurate descriptions of the clusters.
3. **Simple Cluster Definitions:** We have noticed in our experiments that users have a

hard time interpreting the clusters when the clustering algorithm has a complex and non-intuitive definition of a cluster. For example, many clustering algorithms describe a cluster by a long vector of words (the cluster's centroid), and define cluster membership by some similarity threshold between a document and the centroid. Users often have a hard time understanding why certain documents were placed together or what is the "essence" of a certain cluster. We feel that a simpler cluster definition will help users determine what clusters are of interest faster and with more confidence.

These characteristics of the desired clustering algorithm led us to investigate the use of suffix trees for clustering and to develop a novel clustering algorithm – Suffix Tree Clustering.

3.3 Suffix Trees

A suffix tree is a data structure that admits efficient string matching and querying. Suffix trees have been studied and used extensively, and have been applied to fundamental string problems such as finding the longest repeated substring [Weiner, 73], approximate string matches [Landau and Vishkin, 89], string comparisons [Ehrenfeucht and Haussler, 86], and text compression [Rodeh et al., 81]. Following, we describe the suffix tree data structure – its definition, construction algorithms and main characteristics.

3.3.1 Definition

The following description of suffix tree takes after Dan Gusfield's highly recommended book on strings, trees and sequences [Gusfield, 97]. One major difference is that we treat documents as sequences of words, not characters.

A suffix tree of a string is simply a *compact trie* of all the suffixes of that string. In more precise terms:

> **Definition:** A suffix tree *T* for an *m*-word string *S* is a rooted directed tree with exactly m leaves numbered 1 to *m*. Each internal node, other than the root, has at least two children and each edge is labeled with a nonempty sub-string of words of *S*. No two edges out of a node can have edge labels beginning with the same word. The key feature of the suffix tree is that for any leaf *i*, the concatenation of the edge labels on the path from the root to leaf *i* exactly spells out the suffix of *S* that starts at position *i*, that is it spells out *S*[*i*..*m*].

In cases where one suffix of *S* matches a prefix of another suffix of *S* then no suffix tree obeying the above definition is possible since the path for the first suffix would not end at a leaf. To avoid this, we assume the last word of *S* does not appear anywhere else in the string. This prevents any suffix from being a prefix to another suffix. To achieve this we can add a *terminating character*, which is not in the language that *S* is taken from, to the end of *S*.

> **Definition:** The *label of a node* in the tree is defined as the concatenation, in order, of the sub-strings labeling the edges of the path from the root to that node.

Figure 3.1 illustrates the suffix tree of the string *"I know you know I know"*. Internal nodes are marked as circles, leaves as rectangles. There are six leaves in this example, numbered from 1 to 6. The terminating character is not shown in this diagram.

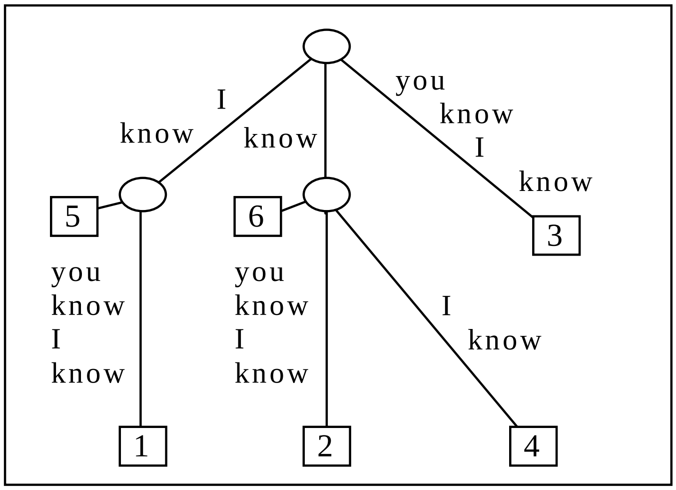


Figure 3.1: Example of a suffix tree

The suffix tree of the string *"I know you know I know"*. There are six leaves in this example, marked as rectangles and numbered from 1 to 6. The terminating characters are not shown in this diagram.

In a similar manner, a suffix tree of a set of strings, called a *generalized* suffix tree, is a compact trie of all the suffixes of all the strings in the set:

> **Definition:** A *generalized* suffix tree T for a set S of n strings S_n, each of length m_n, is a rooted directed tree with exactly Σm_n leaves marked by a two number tuple (k,l) where k ranges from 1 to n and l ranges from 1 to m_k. Each internal node, other than the root, has at least two children and each edge is labeled with a nonempty sub-string of words of a string in S. No two edges out of a node can have edge labels beginning with the same word. For any leaf (i,j), the concatenation of the edge labels on the path from the root to leaf $(i.j)$ exactly spells out the suffix of S_i that starts at position j, that is it spells out S_i $[j..m_i]$.

Figure 3.2 is an example of the generalized suffix tree of a set of three strings – *"cat ate cheese"*, *"mouse ate cheese too"* and *"cat ate mouse too"*. The internal nodes of the suffix tree are drawn as circles, and are labeled *a* through *f* for further reference. Leaves are drawn as rectangles. The first number in each rectangle indicates the string from which that suffix originated; the second number represents the position in that string where the suffix starts. Each string is considered as having a unique terminating character, which is not shown in this diagram.

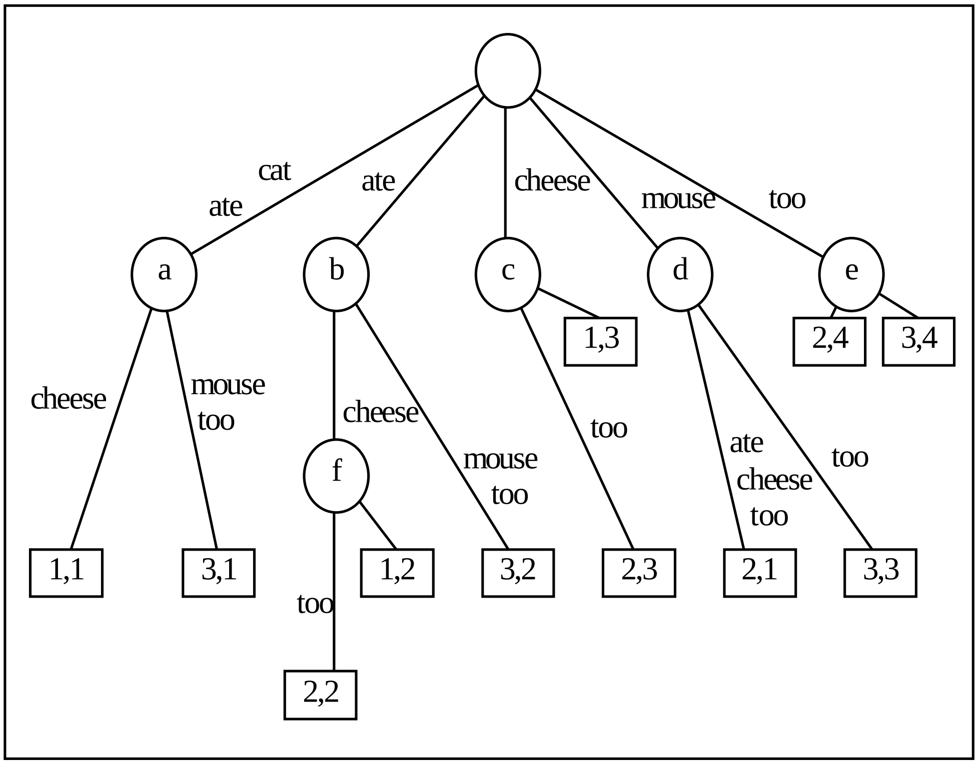


Figure 3.2: Example of a generalized suffix tree

The generalized suffix tree of the three strings *"cat ate cheese"*, *"mouse ate cheese too"* and *"cat ate mouse too"*. The internal nodes of the suffix tree are drawn as circles, and are labeled *a* through *f* for further reference. There are 11 leaves in this example (the sum of the lengths of all the strings) drawn as rectangles. The first number in each rectangle indicates the string from which that suffix originated; the second number represents the position in that string where the suffix starts. The terminating characters are not shown in this diagram.

3.3.2 Construction Algorithms

The naïve, straightforward method to build a suffix tree for a string S of length m takes $O(m^2)$ time. The naïve method first enters a single edge for the suffix $S[1..m]$ into the tree. Then it successively enters the suffix $S[i..m]$ into the growing tree for i increasing from 2 to m. The details of this construction method are presented in Figure 3.3.

1. Let N_i denote the intermediate tree that encodes all the suffixes from 1 to i.
2. Tree N_1 consists of a single edge between the root of the tree and a leaf labeled 1. The edge is labeled with the string S.
3. Tree N_{i+1} is constructed from N_i as follows:
   3.1. Starting at the root of N_i the algorithm finds the longest path from the root whose label matches a prefix of $S[i+1..m]$. This path is found by successively comparing and matching words in suffix $S[i+1..m]$ to words along a unique path from the root, until no further matches are possible.
   3.2. When no further matches are possible, the algorithm is either at a node, say w, or it is in a middle of an edge.
   3.3. If it is in the middle of an edge, say (u,v), then it breaks (u,v) into two edges by inserting a new node w just after the last word on the edge whose label matches a prefix of the suffix $S[i+1..m]$.
   3.4. In either case, the algorithm creates a new edge $(w,i+1)$ running from w to a new leaf labeled $i+1$, and labels the new edge with the unmatched part of suffix $S[i+1..m]$.

Figure 3.3: The naïve suffix tree construction method

The naïve suffix tree construction method is quadratic in the length of the string. To build the suffix tree of string S of length m, it first enters a single edge for the suffix $S[1..m]$ into the tree. Then it successively enters the suffix $S[i..m]$ into the growing tree for i increasing from 2 to m.

Several linear time algorithms for constructing suffix trees exist [Weiner, 73; McCreight, 76; Ukkonen 95]. To be precise, these algorithms also exhibit a time dependency on the size of the vocabulary (or the alphabet when dealing with character-based trees): they actually have a time bound of $O(m * min(log |L|, log m))$ where m is the string's length and $|L|$ is the size of the language.

Weiner's algorithm has larger space requirements than the other two. McCreight's and Ukkonen's algorithms have the same space requirements and can in fact be seen as disguised versions of each other. Ukkonen's algorithm, however, is simpler and has a potentially useful online characteristic. While McCreight's algorithm requires the whole string to be present before starting, Ukkonen's method can process each character in order (*e.g.*, as they are brought to memory from disk or network). We will not be able to present these algorithms here, and will restrict ourselves to a very high level description of Ukkonen's algorithm.

Ukkonen's algorithm constructs a series of *implicit* suffix trees, the last of which is converted into a true suffix tree. An implicit suffix tree is a simplified suffix tree without the addition of a terminating character to the string and without inserting suffixes that match the prefix of other suffixes. The algorithm is divided into m phases. In each phase it constructs an implicit suffix tree I_i for prefix $S[1..i]$ of S (see Figure 3.4 for a high level description). Using several observations and implementation tricks, each of these phases can be done in constant time. The most important element in this acceleration is the use of *suffix links*, which link internal nodes in the tree and provide an efficient way of traversing the suffix tree (namely step 2.1.1 of the outline of the algorithm as presented in Figure 3.4).

1. Construct implicit suffix tree I_1.
2. For i from 2 to m construct I_i from I_{i-1} :
   2.1. For j from 1 to i:
      2.1.1. Find the end of the path from the root labeled $S[j..i-1]$ in the current tree.
      2.1.2. If needed, extend that path by adding character $S(i)$, thus assuring that string $S[j..i]$ is in the tree.

Figure 3.4: Ukkonen's construction algorithm

A high level description of Ukkonen's linear time construction algorithm.

To sum, suffix trees can be constructed in linear time, and furthermore, Ukkonen's algorithm allows this to be done online, as the strings are read into memory. One caveat has to be mentioned: for large trees, paging can be a serious problem because the trees do not have nice locality properties. Indeed, by design, suffix links allow the algorithm to move quickly from one part of the tree to a distant part of the tree. This is great for worse case time analysis but is horrible for paging if the tree does not fit entirely in memory. Consequently, great efforts are often spent to reduce the space requirements of suffix trees. With this in mind, a new data structure, called a *suffix array*, was proposed [Manber and Myers, 93]. Suffix arrays are very space efficient and can be used to solve many string-matching problems almost as efficiently as suffix trees.

3.4 The STC Algorithm

The STC algorithm is based on identifying phrases that are common to groups of documents in the document set. Let's first define what we mean by a phrase:

> **Definition:** A *phrase* in our context is an ordered sequence of one or more words that does not cross a *phrase boundary*. Phrase boundaries are inserted between words when our document-parsing module identifies specific syntactic tokens. These tokens can be punctuation marks (*e.g.*, the characters . (period), , (comma), ; (semicolon), ? (question mark) etc.) or HTML tags (*e.g.*, <p>,
, <li>, <td> etc.). We borrow the term *sentence* to refer to sequences of words between two phrase boundaries (the start and end of the document are also considered phrase boundaries). Note that a phrase can be of any length

We do not allow phrases to cross phrase boundaries as for two reasons: (1) Phrase boundaries usually mark some topical shift, and we have observed that most "interesting" phrases do not cross these boundaries. (2) As will be explained later, this reduces the cost of building our suffix tree, and in some cases (*e.g.*, in the online version of the algorithm) it is critical in order to maintain the linear time complexity.

We also need to define a phrase cluster:

> **Definition:** A *phrase cluster* is a phrase that is shared by at least two documents, and the group of documents that contain the phrase. A *maximal* phrase cluster is a phrase cluster whose phrase cannot be extended by any word in the language without changing (reducing) the group of documents that contain it.

The STC algorithm has three logical steps, which will be detailed in the following sections:

1. **Document Parsing**, in which each document is transformed into a sequence of words and phrase boundaries are identified.
2. **Phrase Cluster Identification**, in which a suffix tree is used to find all maximal phrase clusters. These phrase clusters are scored and the highest scoring ones are selected for further consideration.

3. **Phrase Cluster Merging**, in which the selected phrase clusters are merged into clusters, based on the overlap of their document sets.

In the early stages of this research, phrase clusters were not merged, and a subset of them was presented as clusters [Zamir et al., 97]. The full details of the STC algorithm, as described below, were first published in [Zamir and Etzioni, 98].

3.4.1 Document Parsing

In the first step of the algorithm, each document is transformed into a sequence of words and phrase boundaries are identified. Phrase boundaries are marked by a unique token. Text within parenthesis or quotes is treated as an independent sentence and is placed at the end of the sentence that it appeared in.

Non-word tokens, such as numbers, HTML tags and punctuation are stripped. The remaining words are stemmed using a “light” stemming algorithm which was adapted from the Porter stemmer [Porter, 1980]. We essentially retained only the first component in the Porter stemmer – to reducing plurals to singular. Stemming has been shown to slightly increase the performance (typically recall) of IR systems [Hull, 96], but at the cost of increasing the processing time and reducing the comprehensibility of the text. We chose a “light” algorithm in an attempt to find the “golden path” in this tradeoff.

3.4.2 Phrase Cluster Identification

In this step, the STC algorithm identifies *all* maximal phrase clusters. This is done efficiently using a suffix tree. The identification of phrase clusters can be viewed as the creation of an inverted index of phrases for our document collection. The phrase clusters are scored and the highest scoring ones are selected for further consideration.

We create a generalized suffix tree from all the *sentences* (as defined above – sequence of words between phrase boundaries) of all the documents in our document set. Each leaf is marked with a sentence identifier that also identifies which document it belongs to.

The key feature of the suffix tree is that each internal node v represents a group of documents and a phrase that is common to all of them. The label v_p of internal node v is the common phrase. All the leaves in the sub-tree of v correspond to sentences that have suffixes that start with the phrase v_p. Therefore, the group of documents containing v_p can be determined from these leaves (actually, we can also determine how many times the phrase v_p appears in each document and where).

This leads us to the following observation: every possible *maximal* phrase cluster corresponds to an internal node in our suffix tree. By this we mean that the phrase of the phrase cluster equals the label of the node, and the set of documents of the phrase cluster equals the set of documents designated by the leaves in that node's sub-tree. To see why this is true take a maximal phrase cluster m with phrase m_p. Phrase cluster m contains at least two documents, say i and j, that contain m_p. Therefore documents i and j contain sentences which have suffixes that contain m_p. That means that there exists a path in the suffix tree, starting from the root, whose label starts with the phrase m_p. As there are two sentences (from different documents) that share this phrase, there must be an internal node u on this path, whose label u_p either equals m_p or else is the first label from the root that has m_p as a prefix. The phrase cluster corresponding to node u must include all the documents that contain the phrase m_p, therefore as m is a maximal phrase cluster, m_p must be equal to u_p otherwise a word could have been added to m_p (the next word in u_p after the prefix m_p) without decreasing the number of documents in the phrase cluster.

The reverse is not true. First, an internal node in the suffix tree might not correspond to a phrase cluster as all the leaves in its sub-tree might originate from suffixes of sentences from a single document. An internal node that has leaves in its sub-tree from at least two different documents does correspond to a phrase cluster, but not necessarily to a maximal phrase cluster. This could happen if the phrase v_p of node v is found in all the documents it appears in as a prefix of longer phrase, say v_px, with x being some word, but v_p also appears in at least one of these documents in itself (*i.e.*, ending a sentence or followed by a word other than x). In this case, in the phrase cluster corresponding to node

v, the word x could be added to the phrase v_p without changing it document set, thus it is not maximal.

We can however state a weaker claim: an internal node for which all the leaves in its sub-tree originate from different documents does correspond to a maximal phrase cluster. This is true because if such an internal node v did not correspond to a maximal phrase cluster, then it would necessarily have a child whose set of leaves (in its sub-tree) equals the set of leaves of node v. This would mean that v has a single child, which is contrary to the definition of a suffix tree.

Figure 3.2 presented the suffix tree of the three strings – *"cat ate cheese"*, *"mouse ate cheese too"* and *"cat ate mouse too"*. The internal nodes in the suffix tree in Figure 3.2 were labeled *a* through *f* for further reference. Table 3.1 lists these six internal nodes and their corresponding phrase clusters.

Table 3.1: Internal suffix tree nodes as phrase clusters

Node	***Phrase***	***Documents***
a	cat ate	1,3
b	ate	1,2,3
c	cheese	1,2
d	mouse	2,3
e	too	2,3
f	ate cheese	1,2

The six internal nodes from the example shown in Figure 3.2 and their corresponding phrase clusters.

After creating the suffix tree, we determine which nodes correspond to the maximal phrase clusters, and assign to each of these a score that is a function of the number of documents it contains, and its phrase. This is done in a single traversal of the tree. The score of a phrase cluster attempts to estimate its usefulness for our clustering task. The score $s(m)$ of maximal phrase cluster m with phrase m_p is given by:

$$s(m) = |m| \cdot f(|m_p|) \cdot \Sigma\ \mathrm{tfidf}(w_i)$$

where $|m|$ is the number of documents in phrase cluster m, w_i are the words in m_p, tfidf(w_i) is a score we calculate for each word in m_p, and $|m_p|$ is the number of words in m_p that are not stop-words (*i.e.*, the effective length of the phrase). The function f penalizes single word phrases, is linear for phrase that are two to six words long, and becomes constant for longer phrases.

To determine stopped words, we maintain a standard stoplist containing common English words (*e.g.*, “the”, “there”, “their”, “thus” and “then”). Words appearing in too few (3 or less) or too many (more than 40%) documents are also stopped (this often includes the query terms). We also obtained frequency statistics from a sample of 1GB (~180,000 documents) of Web documents (the BASE1 collection, a uniform sub-collection of the 100GB VLC2 collection currently being used in TREC, consisting of Web data spidered by the Internet Archive in early 1997; this collection was constructd by the WAR/AcSys of the Australian National University). Words appearing in more than 5% of the documents in this sample (*e.g.*, “back”, “java”, “netscape” and “mail”) are also stopped.

Term frequency – inverse document frequency (tfidf) is a commonly used IR technique for assigning weights to individual words (terms) [Salton and Buckley, 88]. The rationale is that the more frequent the word is in a document (the term frequency portion) the more important it is in representing the document’s *aboutness*. Likewise, the less frequent the word is in the whole document collection (the inverse document frequency portion) the more likely it is to be a good differentiator between documents, and thus more important to the IR task. We calculate the tfidf(w_i,d) score of word w_i in document d using the following formula [Salton and Buckley, 88]:

$$\mathrm{tfidf}(w_i,d) = (1+\log(\mathrm{tf}(w_i,d))) \cdot \log(1 + \mathrm{N}/\mathrm{df}(w_i))$$

where tf(w_i,d) is the number of occurrences of word w_i in document d, N is the total number of documents in our document set and df(w_i) is the number of documents that term w_i appears in. However, we have found that the inverse document frequency component appears to be more accurate when we use the statistics of the 1G BASE1 collection, rather than of the retrieved document set. Therefore N will equal the number of documents in the BASE1 collection (180,000), and df(w_i) is the number of documents in the BASE1 collection that term w_i appears in.

Finally, we select only the k highest scoring phrase clusters (we take k to be 500 in our experiments). This selection is designed to keep the cost of the next step constant, but we have found that it also has another important advantage: it prevents the next step from being influenced by low scoring, and thus presumably less informative, phrase clusters.

3.4.3 Phrase Cluster Merging

In the previous step, we have identified groups of documents that share phrases. However, documents may share more than one phrase. Therefore, the document sets of distinct phrase clusters may overlap and may even be identical. To avoid the proliferation of nearly identical clusters, the third step of the STC algorithm merges phrase clusters with a high overlap in their document sets (phrases are not considered in this step).

We define a binary similarity measure between phrase clusters based on the overlap of their document sets. Given two phrase clusters m_i and m_j, with sizes $|m_i|$ and $|m_j|$ respectively, and $|m_i \cap m_j|$ representing the number of documents common to both phrase clusters, we define the similarity $sim(m_i,m_j)$ of m_i and m_j to be:

$$sim(m_i,m_j) = 1 \quad \text{if} \quad |m_i \cap m_j| / |m_i| > \alpha \text{ and } |m_i \cap m_j| / |m_j| > \alpha$$

$$sim(m_i,m_j) = 0 \quad \text{otherwise}$$

where α is a constant between 0 and 1 (we typically use $\alpha = 0.6$).

Next, we look at the *phrase cluster graph*, where nodes are phrase clusters, and two nodes are connected if and only if the two phrase clusters have a similarity of 1. A cluster is defined as being a connected component in the phrase cluster graph. We call these *merged clusters*. Each merged cluster contains the union of the documents of all its phrase clusters. Figure 3.5 illustrates the phrase cluster graphs of the six phrase clusters from Table 3.1, for α values of 0.6 and 0.7. Table 3.2 lists the final merged clusters identified from the based cluster graphs of Figure 3.5.

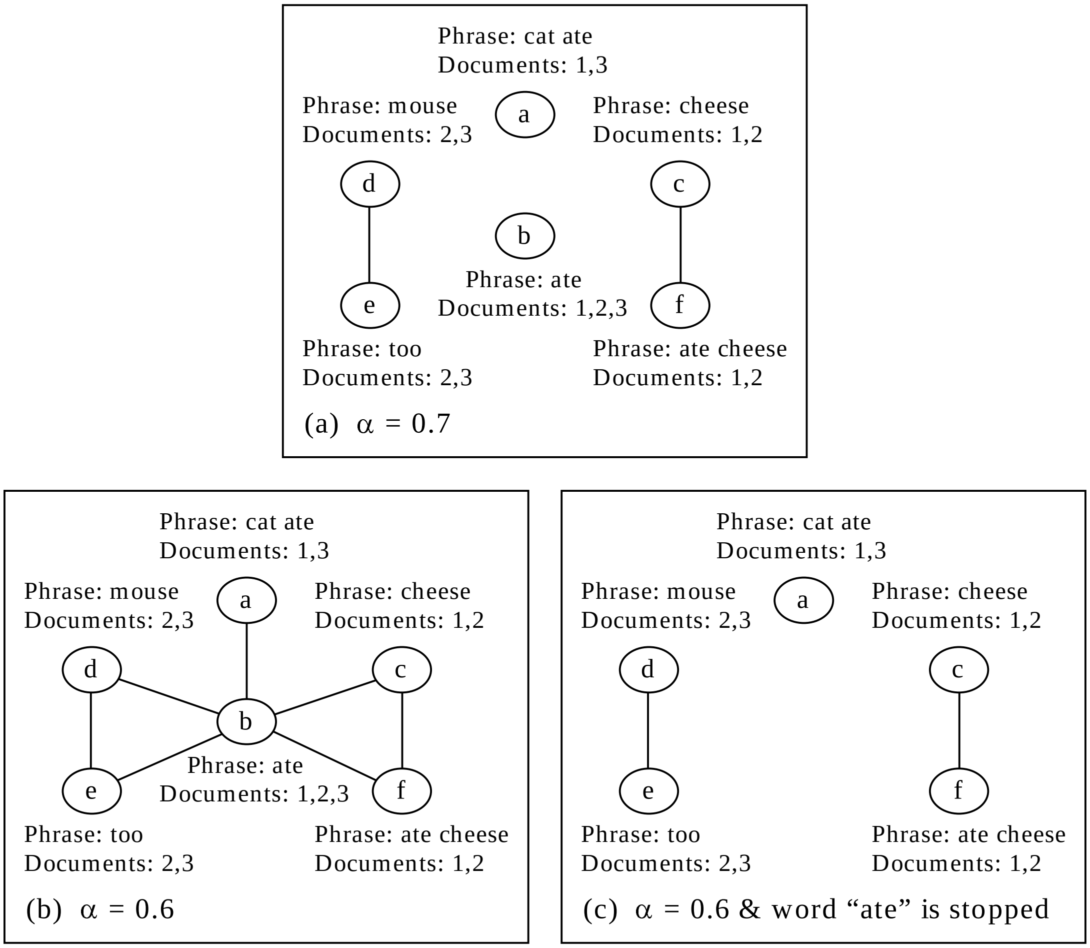


Figure 3.5: The phrase cluster graph

The phrase cluster graphs of the six phrase clusters from Table 3.1. **(a)** for $\alpha = 0.7$ there are four connected components is the graph, representing four merged clusters. **(b)** for $\alpha = 0.6$ there is a single connected component is the graph, representing one merged cluster. **(c)** If the word *ate* had been in our stoplist, the phrase cluster *b* would have been discarded as it would have had a score of 0, and for $\alpha = 0.6$ we would have had three connected components in the graph, representing three merged clusters.

Table 3.2: Merged clusters as connected components in the phrase cluster graph

Figure	Cluster Number	Phrase clusters	Documents
(a)	1	a	1,3
	2	b	1,2,3
	3	d,e	2,3
	4	c,f	1,2
(b)	1	a,b,c,d,e,f,g	1,2,3
(c)	1	a	1,3
	2	d,e	2,3
	3	c,f	1,2

The final merged clusters identified from the phrase cluster graphs of Figure 3.5.

In essence, In this step we are clustering the phrase clusters using the equivalent of a single-link clustering algorithm, where a predetermined minimal similarity between phrase clusters serves as the halting criterion. We do not encounter the undesired chaining effect of single-link clustering because in the realm of phrase clusters we typically find only small connected components.

We process the phrase clusters one at a time, in descending order, based on their scores. For each phrase cluster, we check its similarity to all phrase clusters already processed, and update the phrase cluster graph and the resulting clusters accordingly. Thus we can produce meaningful results even if an impatient user halts the algorithm in the middle of this phase.

As mentioned before, in this step we process only the *k* highest scoring phrase clusters. We also have a minimum phrase cluster score threshold and in addition discard the lower scoring 20% of the phrase clusters. This is important to keep the cost of this step constant, but it is also important in preventing this step from being influenced by low scoring, and thus presumably less informative, phrase clusters.

The final merged clusters are scored and sorted based on the scores of their phrase clusters and their overlap. Currently we use a simple scoring method: the score of a merged cluster is equal to the sum of the scores of its phrase clusters. As the final number of merged clusters can vary, we consider (and report) only the top few clusters. Typically, only the top 10-15 clusters are reported.

3.5 Complexity

In this section we will show that the non-incremental version of STC has a linear time complexity. The incremental version of the algorithm is presented and analyzed in the following section.

The cost of the first step, document parsing, is obviously linear with the document set length (the sum of the lengths of all the documents in the clustered document set). If we assume the length of each document to be bound by some maximal length, this step is linear with the number of documents to be clustered.

To construct the suffix tree we use Ukkonen's algorithm, which is also linear with the document set length. In one depth-first traversal of the suffix tree we can determine which nodes correspond to maximal phrase clusters, calculate their scores, and select the top *k* phrase clusters. To determine maximal phrase clusters we must calculate the number of documents represented in the sub-tree of each node. For each node we determine the set of documents that is represented in its sub-tree by performing a union of the sets of all of its children. This process has the potential to be quadratic in the number of documents. But in our domain, sentences are very rarely more than twenty words long, and any longer sentences are split into sections of twenty words in length. This ensures us that the longest path from the root to any leaf in the suffix tree includes no more than twenty internal nodes. This, in turn assures that calculating the number of documents represented the sub-tree of each node is linear.

The third step of the algorithm is quadratic in the number of phrase clusters being compared, but as we bound this by a constant *k*, we can treat this step as being of constant time. Thus, the overall time complexity of STC is linear with regard to the document set length, or if we assume the length of the documents is bounded by some maximal length, STC is linear with the number of documents to be clustered.

3.6 Incremental STC

The STC algorithm can also be performed incrementally, but a few changes are necessary. First, as each document is added to the suffix tree we need to update the score of each internal node that is effected by this update. Affected nodes include internal nodes created by this insertion, but also all internal nodes that are ancestors of new leaves in the tree. Ukkonen's construction algorithm does not allow us to do this easily as suffix links are used to traverse the tree and leaves are created without necessarily "passing" through all their ancestors. We therefore use the naïve suffix tree construction algorithms. As we insert sentences into the suffix tree we update the scores of the nodes we encounter, as well as determine which correspond to maximal phrase clusters (here, we do not need to determine the set of documents corresponding to each nodes to identify maximal phrase clusters). This construction algorithm is quadratic in the length of the longest string to be inserted (the longest sentence). However, as all our sentences are bounded by a constant length, the time to insert each sentence using the naïve method can be taken as constant, and therefore the time to construct the full suffix tree is linear with the document set length.

As each sentence is inserted, we keep a list of all nodes that are updated (or created). The number of nodes updated can be quadratic with the length of the sentence, and therefore can be treated as constant. Next, we have to update the corresponding phrase clusters (only if they are among the top k phrase clusters) and recalculate the similarity of these phrase clusters to the rest of the phrase clusters. Finally, we have to check if the changes in the phrase cluster graph result in any changes to the final clusters.

In the worst case, recalculating the similarities between the phrase clusters can take $O(k^2)$, so the whole algorithm will take $O(lk^2)$ with l being the document set length and k

the number of phrase clusters we maintain for step three of the algorithm. However, with some engineering tricks this step can be made much faster than its worst case bound. It should also be noted that there is no need to actually run this step after every document is received – the phrase cluster graph can be updated whenever the system has free time as it is waiting for more data. This will obviously speed the total processing time, but will violate the any-time characteristic of the incremental algorithm.

The execution of the incremental version of STC, although linear, is obviously much slower than the offline version (as is often the case). The incremental version would therefore be useful when there is a considerable time delay (on the order of the clustering time) for loading the documents into memory, such as when using a slow modem to get the documents off the Web.

3.7 STC Characteristics

We have presented the STC algorithm and have shown it to be linear in the document set length, as well as incremental, if needed. These are crucial for any interactive clustering system. We now focus on two additional characteristics of the algorithm: being phrase-based and generating overlapping clusters.

As mentioned previously, most clustering algorithms treat a document as a set of words. STC identifies common phrases in the documents and uses them as the basis for clustering. This can improve the quality of the clusters by utilizing more information present in the text of the documents, and is useful in constructing concise and accurate descriptions of the clusters. We have noticed in our experiments that users often had a hard time interpreting the clusters when the clustering algorithm has a non-intuitive definition of a cluster. STC used a simple cluster definition: all documents that contain at least one of the cluster's (hopefully related) phrases are members of that cluster. We believe this to be a much more intuitive definition for users to understand than the common "dot product with centroid" membership criterion.

Allowing a document to appear in more than one cluster acknowledges that documents are complex objects that may be grouped into multiple potentially overlapping, but internally coherent, groups. This is actually the reason many IR system use some form of dot-product document similarity measure (as opposed to Euclidean distance, for example): it allows a document to be similar to multiple distinct documents or centroids that could in turn be very dissimilar from each other.

In STC, as documents may share more than one phrase with other documents, each document might appear in a number of phrase clusters. Therefore, a document can appear in more than one cluster. Note that the overlap between clusters cannot be too high, otherwise they would have been merged into a single cluster.

We believe that this feature is highly advantageous to the user in terms of understanding the clusters. But there is another advantage to this feature – if a document is erroneously placed in the wrong cluster (*e.g.*, because of a fluke high similarity with unrelated documents) the effects of this are not as bad as in algorithms with non-overlapping clusters. This is because in STC the document will also (hopefully) be placed in its "correct" cluster. This will result in a false-positive error but not necessarily in its corresponding false-negative error.

The STC algorithm does not require the user to specify the required number of clusters, as do many of the clustering algorithms. It does require, on the other hand, the specification of the threshold α used to determine the similarity between phrase clusters. However, we have found that the performance of STC is not very sensitive to this threshold, unlike hierarchical algorithms for example that show extreme sensitivity to the number of clusters required [Milligan and Cooper, 85].

3.8 Summary

We have introduced the Suffix Tree Clustering algorithm, which generates and describes clusters based on common phrases that are found in the document set. STC uses a suffix tree to efficiently identifying these common phrases. We have shown STC to be linear in the length of the document set, incremental, and to produce overlapping clusters. In the next chapter we shall evaluate the quality of the clusters produced by the STC algorithm.

Chapter 4

EVALUATION OF SUFFIX TREE CLUSTERING

4.1 Introduction

We begin this chapter by empirically evaluating the performance of the STC algorithm and comparing it to other commonly used (vector-based) clustering algorithms. There are three basic questions we address: (1) Does the STC algorithm produce clusters of higher quality? (2) If so, can we identify which of its characteristics contribute to its quality? (3) Is STC fast?

We start by describe the three document collections used in our experiments: two collections of Web Search Results and the OHSUMED medical abstracts collection. In section 4.3 we evaluate STC's "clustering quality". This is done using two quality-evaluation approaches – the IR approach and the merge-then-cluster approach. In the following section we investigate the importance of overlapping clusters and of the use of phrases for STC by evaluating hobbled versions of the STC algorithm without these features. We also investigate whether overlapping clusters can improve the performance of other clustering algorithm.

After evaluating the STC algorithm we explore the question: Will phrases also improve the clustering quality of other clustering algorithms besides STC, and if so why? This question will be addressed in section 4.5. After showing that phrases greatly improve performance across all algorithms we investigate where the advantage of using phrases comes from. Is it simply their being multiword features or is the adjacency and order information also useful? To address this we investigate a variation of STC that uses frequent sets as multiword features instead of phrases. We also compare n-grams to suffix tree phrases to evaluate the added importance of using long phrases for clustering.

In section 4.6 we examine the execution speed of STC and compare it to other fast clustering algorithms. Finally, we ask whether on the Web, snippet clustering is a reasonable approximation to document clustering. This issue is addressed in section 4.7.

4.2 Document Collections

We used three document collections is our experiment: two Web Search Results (WSR) collections (WSR-SNIP and WSR-DOCS) that we constructed for this research, and a standard medical collection – OHSUMED – that is used extensively in IR research.

As we were interested in the performance of document clustering on *Web search engine results*, we had no choice but to construct such a collection for our experiments. Standard IR collection (e.g., TREC, Reuters-21578, OHSUMED) contain documents that are quite different from typical Web documents, and it has been shown that performance of IR systems can be highly dependent on the collection used.

The WSR collections were constructed by saving the results of different queries to the MetaCrawler search engine [Selberg and Etzioni, 95]. As the MetaCrawler is a *meta* search engine, (*i.e.*, it routes queries to several other search engines and then collates their results), we assume its search results and the snippets it returns are representative of Web search engines. The process was as follows: We first defined 10 queries by specifying their topics (*e.g.*, “black bear attacks”) and their descriptions (*e.g.*, “we are interested in accounts of black bear attacks on humans or information about how to prevent such attacks”). The words appearing in each query's topic field were used as keywords for a Web search using the MetaCrawler search engine. We saved the top 1000 hits from the search engine for each query. We thus generated 10 sets of 1000 snippets from the results of these queries. This is the WSR-SNIP collection. To generate the WSR-DOCS collection, we downloaded the original Web document of each of the snippets in the WSR-SNIP collection (we timed-out this download process after one hour for each query). We thus generated 10 corresponding sets of 1000 Web documents.

Next, we manually assigned a relevance judgment (relevant or not) to the top 200 document in these sets based on their fit to the queries' descriptions. The relevance judgment was based on the textual information present in both the snippet returned from the search engine and the original Web document itself. On average there were 41 relevant documents for each query.

The OHSUMED medical abstracts collection [Hersh et al., 94] was created to assist information retrieval research. It is a clinically oriented MEDLINE subset, consisting of 348,566 references (out of a total of over 7 million), covering all references from 270 medical journals over a five-year period (1987-1991). Each document in OHSUMED contains one or more Medical Subject Heading (MeSH) indexing terms. MeSH comprises the National Library of Medicine's controlled vocabulary used for indexing articles. MeSH organizes its descriptors in a hierarchical structure and includes more than 19,000 main headings. OHSUMED documents have primary and secondary MeSH indexing terms, discriminating between topics that are the focus of the document vs. topics that are briefly mentioned.

In this study we used a subset of the OHSUMED collection. We selected documents from 1990 and 1991, which contain non-empty title and abstract fields (about 20% of the documents have an empty abstract field). Furthermore, we selected only documents that have a primary MeSH index term in the “Cardiovascular” sub-branch of the MeSH hierarchy. This gave us a total of 7061 documents.

For some of the experiments we needed to define a set of disjoint groups of OHSUMED document, each group relating to a specific topic. We created these groups as follows: There are 112 index terms under the “Cardiovascular” term in the MeSH hierarchy. For each of these terms we collected its *document group*: all the 1990-1991 OHSUMED documents that contain that term as a primary index term, but contain no other primary index terms from the “Cardiovascular” sub-branch of the MeSH hierarchy. We discard document groups contained less than 10 or more than 100 documents (thus discarding 22 terms). We also discard document groups whose term is an ancestor (in the MeSH hierarchy) of another selected term. This process ensures that the remaining 84 term document groups contain documents that relate to specific, disjoint topics. Finally, we created 60 document sets by randomly merging 2, 4 or 8 document groups (20 set of each). These sets were used in the “merge-then-cluster” experiments as we could compare the clusters created by the algorithms to the original partition of documents into their respective document groups. These document sets were also used in the IR evaluation methodology by assuming, for each set, that only the documents originating for one term document group are relevant.

4.3 Clustering Quality

In this section, we evaluate the “clustering quality” of STC as compared to other clustering algorithms and to the ranked list presentation of the search results. The clustering algorithms compared are Buckshot, Fractionation, group-average hierarchical clustering (GAVG), k-means, single-pass and STC. The GAVG algorithm was chosen as it is commonly used; the rest were chosen as they are fast enough to be contenders for online clustering. The clustering quality is compared using two quality-evaluation approaches: the IR approach (for all three collections) and the “merge-then-cluster” approach (for the OHSUMED collection).

4.3.1 IR Evaluation of Clustering Quality

The field of IR has typically been concerned with the comparison and evaluation of *ad-hoc* search results. Evaluation metrics were developed and analyzed for this task, typically under the premise that the search results are returned as a ranked list of documents, each judged as being relevant or not to the query at hand. These metrics have been adapted to the evaluation of document clustering algorithm. The next section described the methodology used for such an evaluation, and the following section presents the results of our experiments.

4.3.1.1 Methodology

Search results are typically presented as a ranked list of documents, each judged as being relevant or not to the query at hand. Several metrics are commonly used in IR to evaluate the quality of search results. Among the most common metrics used are precision-recall graphs, the average precision metric and expected search length graphs.

To adapt these metrics to evaluate clustering algorithms we use the resulting clusters to reorder the ranked list of documents returned by the search engine. This typically requires multiple assumptions regarding the system and the user (a user model). For example it is common to assume that the user is able to locate the "best" cluster (the one with the highest density of relevant document) with 100% accuracy [Hearst and Pedersen, 96; Schütze and Silverstein, 97]. This is, needless to say, a false assumption, not only because the user is not perfect, but also because the ability of the system to describe the contents of each cluster is not perfect (and, of course, the whole notion of relevance is in itself problematic [Harter, 96]). Moreover, empirical tests have shown that users fail to select the best cluster about 20% of the time [Hearst and Pedersen, 96]. We believe a more realistic model should be used, in which the probability a user will select a cluster is a function of the density of the relevant documents in that cluster. Nevertheless, for this research we have followed the 100% user-accuracy model.

A common extension to this user model is that users only look at documents from a single cluster (the one-cluster viewed user model) [Hearst and Pedersen, 96; Schütze and Silverstein, 97]. We believe this model to be problematic, as different clustering algorithms tend to produce clusters of different size distributions. This can cause IR metrics such as average precision to be influenced by the size distribution and not by the clustering itself (as we demonstrate in Figure 4.1). It can also produce skewed results as it over penalizes small clusters, even if they have a very high density of relevant documents. We have therefore adapted a modified user model, in which the user looks at only a constant number of documents (starting with the "best" cluster and working her way through subsequent clusters until she has viewed the desired number of documents). In our experiment, we assume the user glances at only 10% of the documents, the remaining 90% of the documents are considered irrelevant (as the user does not view them). Thus, while the one-cluster-viewed model assumes the user picks a single cluster (even if it contains say 50% of the document set), our user model assumes that the user may view more than one cluster, or only a fraction of one in the case where the top cluster contains more than 10% of the documents in the collection. While actual user behavior is quite complex and idiosyncratic, we believe that our methodology provides a better model of user behavior. A user picking a large cluster to investigate first might not scan all the documents in it, while a user picking a small cluster first might proceed to a second cluster once she's done with the first.

When this reordering method is applied to overlapping clusters, one might view the same document more than once. If a document is seen a second time, it should either be ignored (*i.e.*, the user identifies that it has been viewed before) or deemed irrelevant (*i.e.*, the user wastes time on it just like any other irrelevant document without added informational value). We used both these options in our experiments, but the difference between them was marginal (see Figure 4.2). Therefore, unless otherwise noted, we display the average of these two measures.

We use three basic IR metrics in our experiments:

1. **Average Precision:** Average precision is defined as the average of the uninterpolated precision calculated at each relevant document position along the ranked list. As we assume the user views only a subset of the relevant documents, we sum the uninterpolated precision only for relevant document viewed, and divide it by the total number of relevant documents, whether viewed or not (this is after [Hearst and Pedersen, 96]; Figures 4.3 – 4.5).
2. **Precision-Recall Graph:** We present the 11 point interpolated Precision-Recall graphs. Here we assume the user views the clusters in order until 100% recall of all the relevant documents in the collection is attained (Figures 4.6 – 4.8).
3. **Expected Search Length:** ESL is defined as the expected number of irrelevant documents that are viewed before encountering *r* relevant ones [Cooper, 88]. We present the Number-to-View graphs, which plot the ESL as a function of *r* [Dunlop, 97] (Figures 4.9 – 4.11).

We calculated these three metrics for the six clustering algorithms, for the original ranked list (for the WSR collections), and for random ranked lists (for the OHSUMED collection). We also compared the results to a random clustering of the documents. This was meant to provide a base line for comparison, as the reordering of the documents (using external information to choose the best clusters first) can produce results that are superior to the original ranked list even for a random clustering. We used different random clusterings, with different cluster size distributions, and asserted that our metrics are not significantly influenced by this (see Figure 4.2). The results of the random runs were averaged over 100 random clusterings and unless otherwise mentioned, we present the results of the clusters' size distribution that achieved the best score.

Figure 4.1 shows that the one-cluster-viewed user model can be highly effected by the clusters' size distribution. Here we present the average precision for the results of the six clustering algorithms, the original list and three different random clustering size distributions, given the one-cluster-viewed user model. The "random equal" distribution assumes each document has an equal probability of membership in each cluster. The "random 30%" distribution assumes each document has a 30% probability of being in cluster 0 (a single large cluster), and 70% probability of being (equally distributed) in any of the other clusters. The "random 50%" distribution similarly assumes each document has a 50% probability of being in the large cluster. We can see there is a 30% difference in the average precision metric given the one-cluster user model between the different random cluster size distributions.

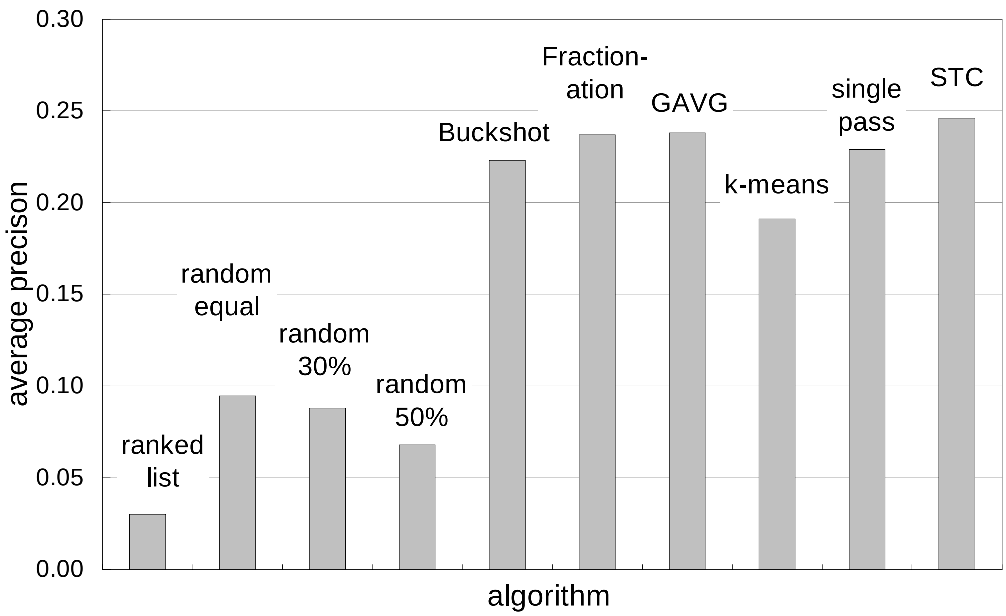
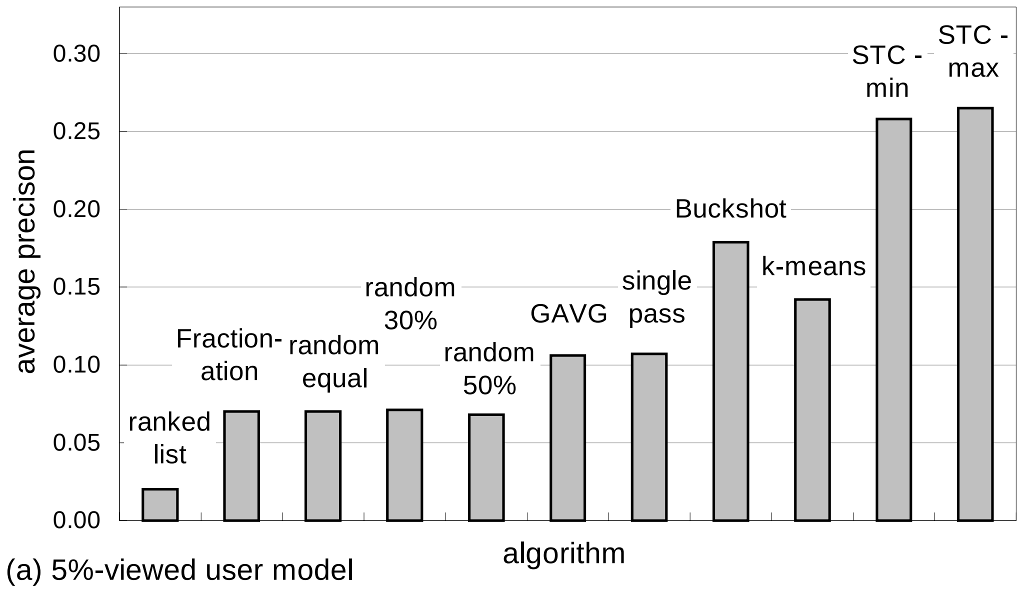


Figure 4.1: Average precision given the one-cluster user model

The average precision for the results of the six clustering algorithms, the original list and three different random clustering size distribution, given the one-cluster user model on the WSR-DOCS collection. The results demonstrate that the one-cluster model can be highly effected the clusters' size distribution. We can see there is a 30% difference in the average precision metric between the different random cluster size distributions.

Figure 4.2 presents the average precision of the different algorithms on the WSR-DOCS collection under different user models in which the user views 5%, 10%, 20% or 100% of all the documents. It can be seen that the relative ranking of the algorithms hardly changes with these different values, and therefore, we have selected the 10%-viewed model as a reasonable model to base our comparison upon. The three random clustering size distributions presented earlier ("random equal", "random 30%" and "random 50%") were also used and show that this evaluation method is very stable under different random cluster size distributions.

We also present two STC evaluations that treat documents viewed multiple times differently. "STC – max" assumes the document should be ignore (*i.e.*, the user notes that it has been viewed before), while “STC – min” assumes the document is irrelevant (*i.e.*, the user wastes viewing time on it just like any other irrelevant document). This distinction is important only when evaluating clustering algorithms that produce overlapping clusters. As can be seen from the figure, the difference between the two is not large (~7%). We actually believe the true “cost” of a document should be somewhere in the middle of these two evaluations, and unless otherwise mentioned, we average these two evaluations in future graphs.



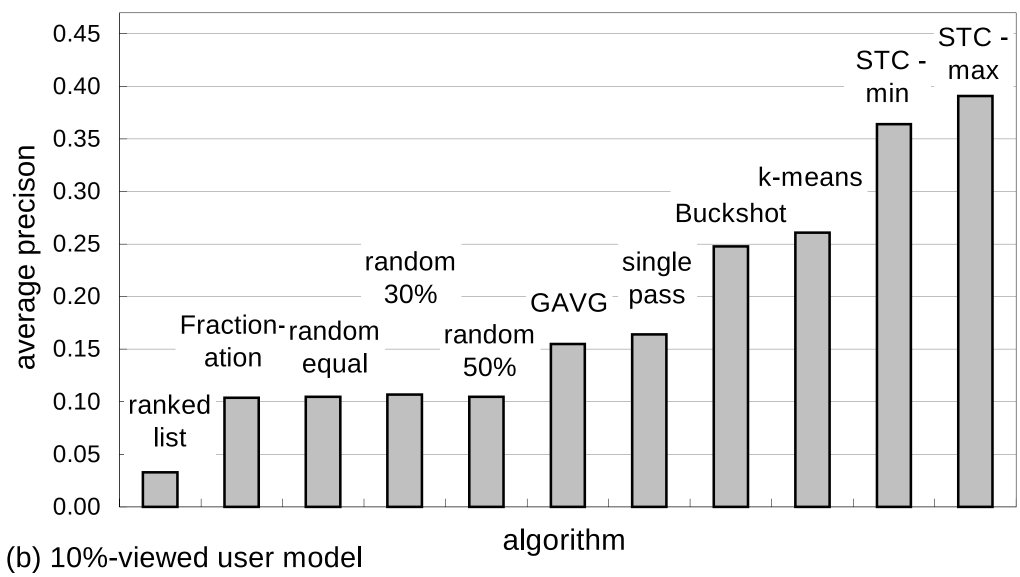
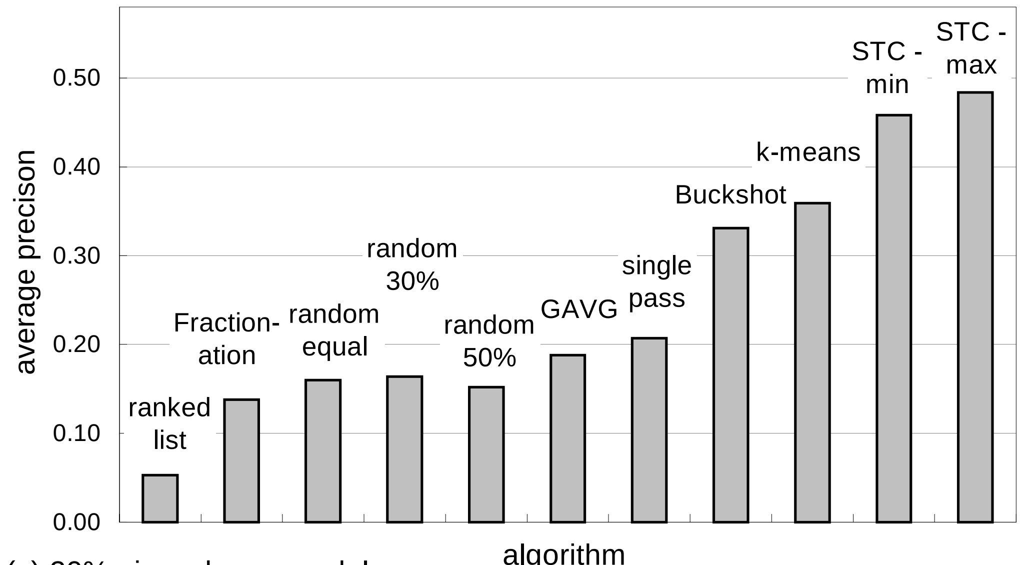
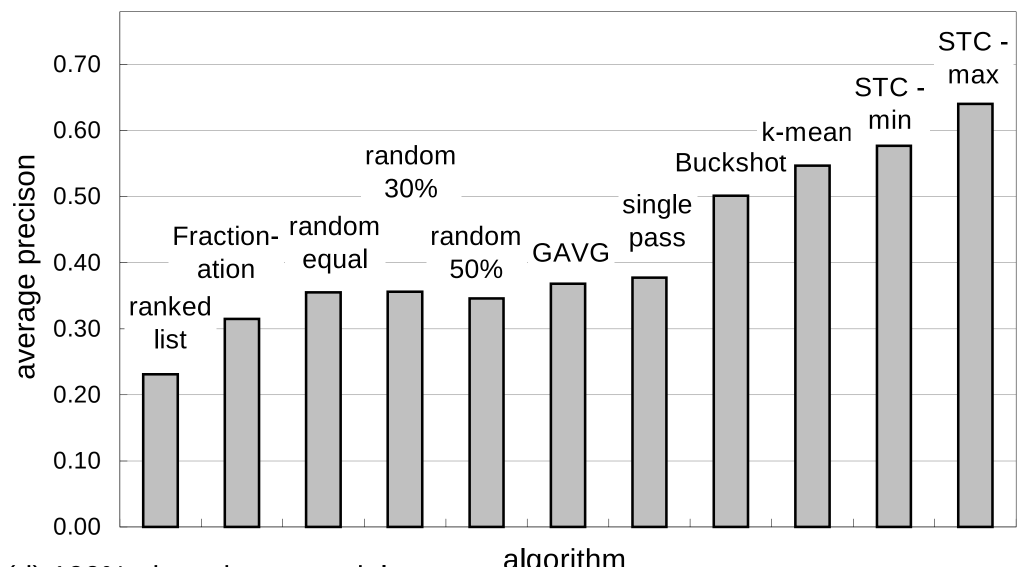


Figure 4.2: Average precision using four different user models

The average precision of the six clustering algorithms, the original list and three random clusterings (with different cluster size distributions), on the WSR-DOCS collection. We compare four user models in which the user views 5%, 10% 20% or 100% of the documents. The first two are shown on this page, the remaining two on the following page. (Continued on following page)



(c) 20%-viewed user model



(d) 100%-viewed user model

Figure 4.2: Average precision using four different user models (cont.).

We compare four user models in which the user views 5%, 10% 20% or 100% of the documents. The relative ranking of the algorithms changes little (actually only two algorithm changed relative positions once – the k-means and the Buckshot algorithms in the 5%-viewed model) among these different user models. These user models are also very stable under different random cluster size distributions.

All algorithms were run to produce the same number of clusters (10 in our experiments). This is necessary to allow a fair comparison of the different algorithms. The algorithms use the same parameter settings wherever relevant (*e.g.*, the minimal cluster size); the rest of the parameters were optimized (separately for each document collection) on a separate data set. The results are averaged over the 10 document sets in the WSR collections and over the 60 document sets of the OHSUMED collection.

4.3.1.2 Results

Figures 4.3 through 4.5 compare the average precision for the results of the six clustering algorithms, the original list and random clustering, given the 10%-viewed user model. Figures 4.3, 4.4 and 4.5 compare the algorithms on the WSR-SNIP, WSR-DOCS and OHSUMED collections respectively. As can be seen from the results, STC outperform all other algorithms, except for the k-means algorithm on the OHSUMED collection. Note that the performance of the algorithms varies greatly across the different document collections.

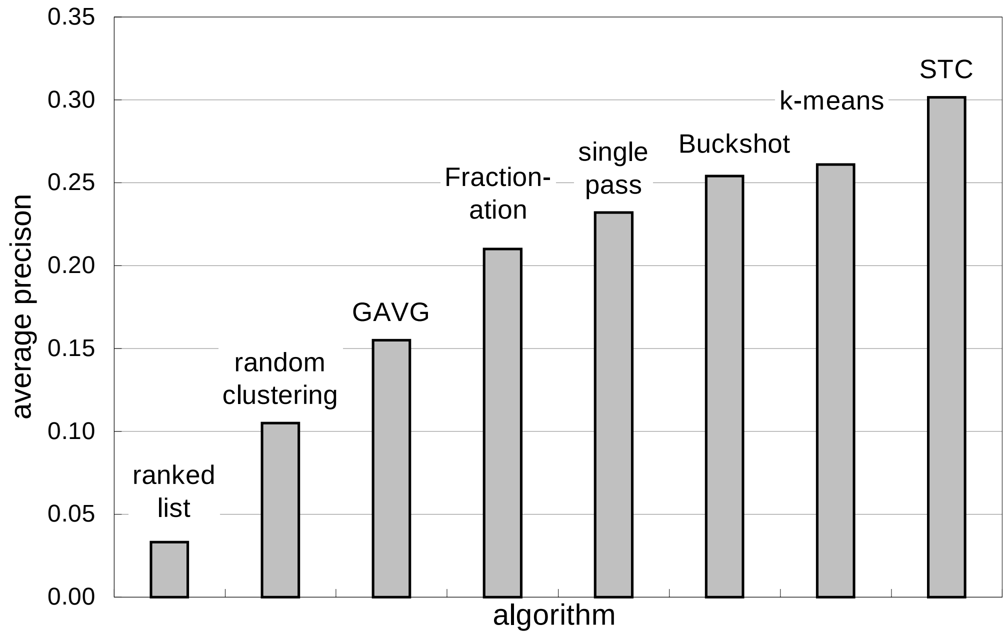


Figure 4.3: Average precision on the WSR-SNIP collection

The average precision of the six clustering algorithms, the original list and random clustering, given the 10%-viewed user model, on the WSR-SNIP collection.

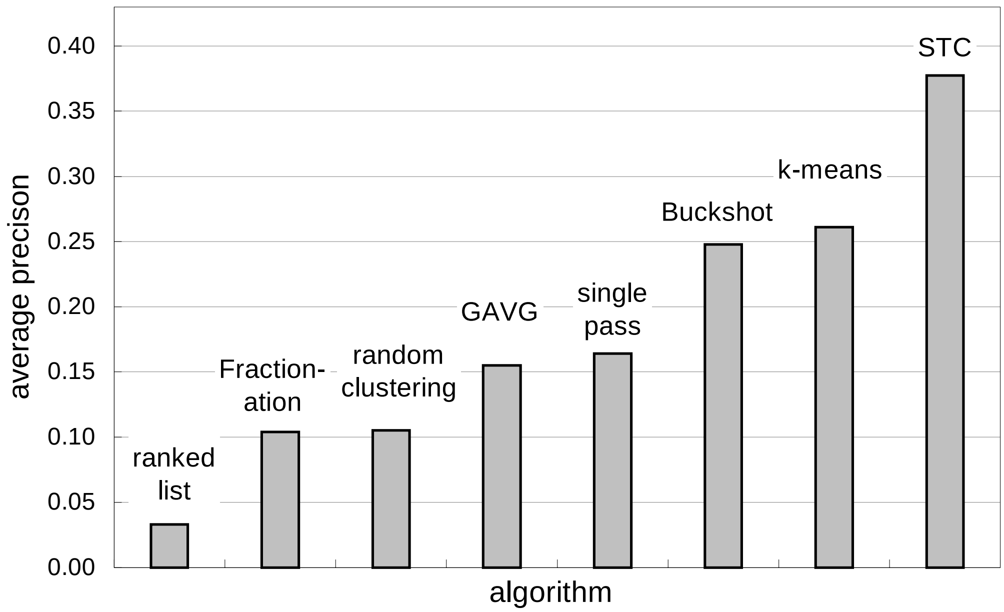


Figure 4.4: Average precision on the WSR-DOCS collection

The average precision of the six clustering algorithms, the original list and random clustering, given the 10%-viewed user model, on the WSR-DOCS collection.

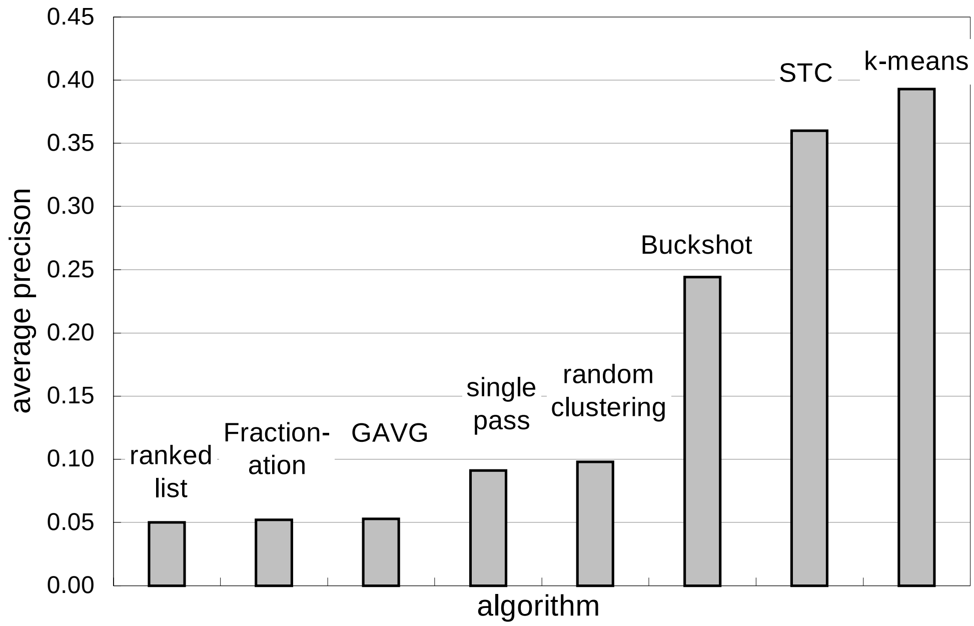


Figure 4.5: Average precision on the OHSUMED collection

The average precision of the six clustering algorithms, the original list and random clustering, given the 10%-viewed user model, on the OHSUMED collection.

Statistical significance tests were performed using the two-tailed paired sign test. We have only 10 document sets in the WSR collections and therefore for two algorithms to be statistically different from each other ($p < 0.05$), one has to outperform the other in 9 out of the 10 runs (and then $p = 0.021$). On the WSR-SNIP collection (Figure 4.3), the difference between STC and single-pass was significant, but the difference to k-means and Buckshot was not. On the WSR-DOCS collection (Figure 4.4), STC did outperform all other algorithms with statistical significance. On the OHSUMED collection (Figure 4.5), the difference between k-means and STC is only marginally significant ($p = .092$); the difference between STC and the rest of the algorithms is significant.

In Figures 4.6, 4.7 and 4.8 we present the precision-recall graphs of the different algorithms for the three text collections. The “STC–min” and the “STC–max” evaluation methods were averaged into a single STC measurement in the interest of clarity, as the difference between them was always quite small (~5%). The three random clusterings (with different size distributions) exhibited practically identical precision-recall graphs and therefore only one is shown.

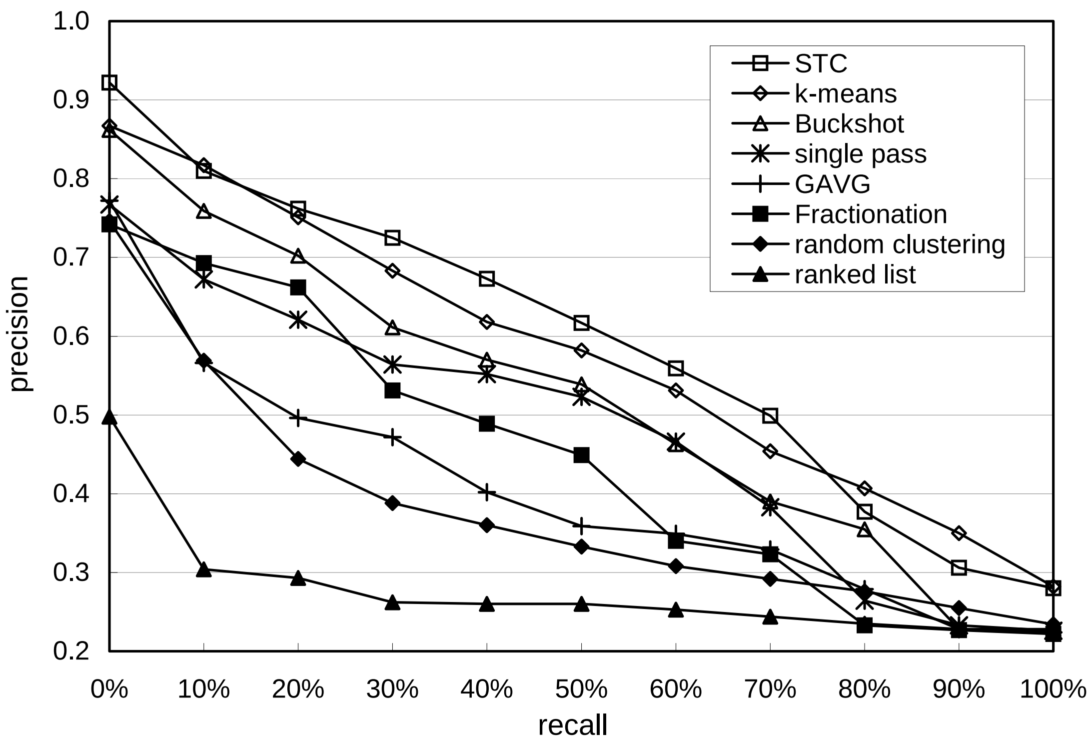


Figure 4.6: Precision-recall graph for the WSR–SNIP collection

The precision-recall graph for the results of the six clustering algorithms, the original ranked list and random clustering for the WSR–SNIP collection. Again, we can see STC outperform the other algorithms, with k-means and Buckshot closely following.

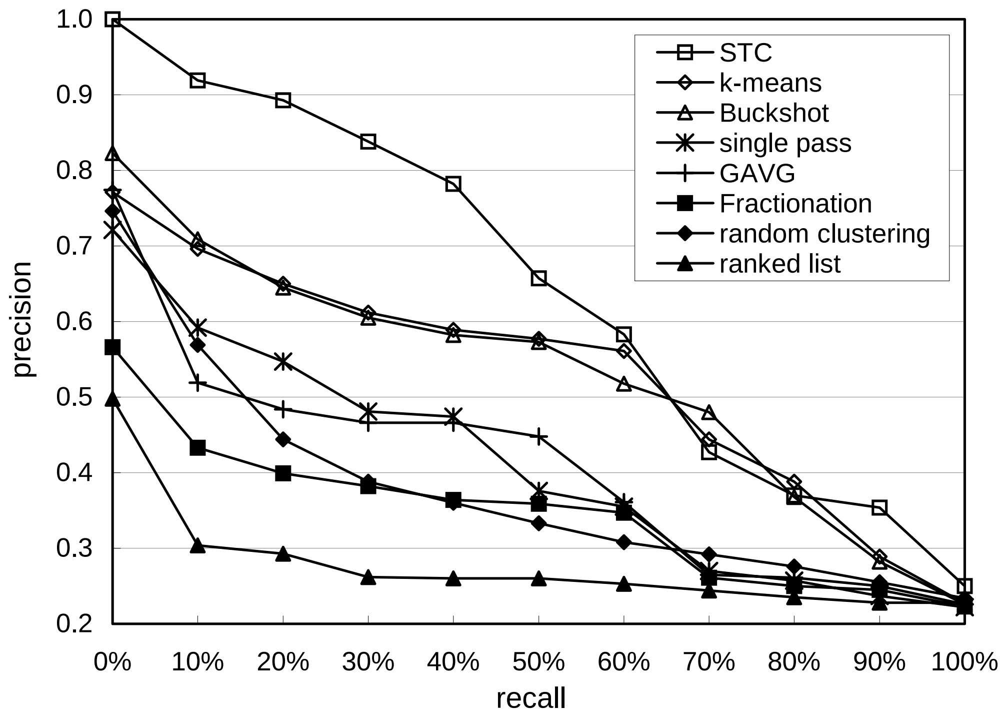


Figure 4.7: Precision-recall graph for the WSR–DOCS collection

The precision-recall graph for the results of the six clustering algorithms, the original ranked list and random clustering for the WSR–DOCS collection. Three clustering algorithms – STC, k-means and Buckshot – seem to perform significantly better than random clustering, and of these STC clearly outperforms the other two.

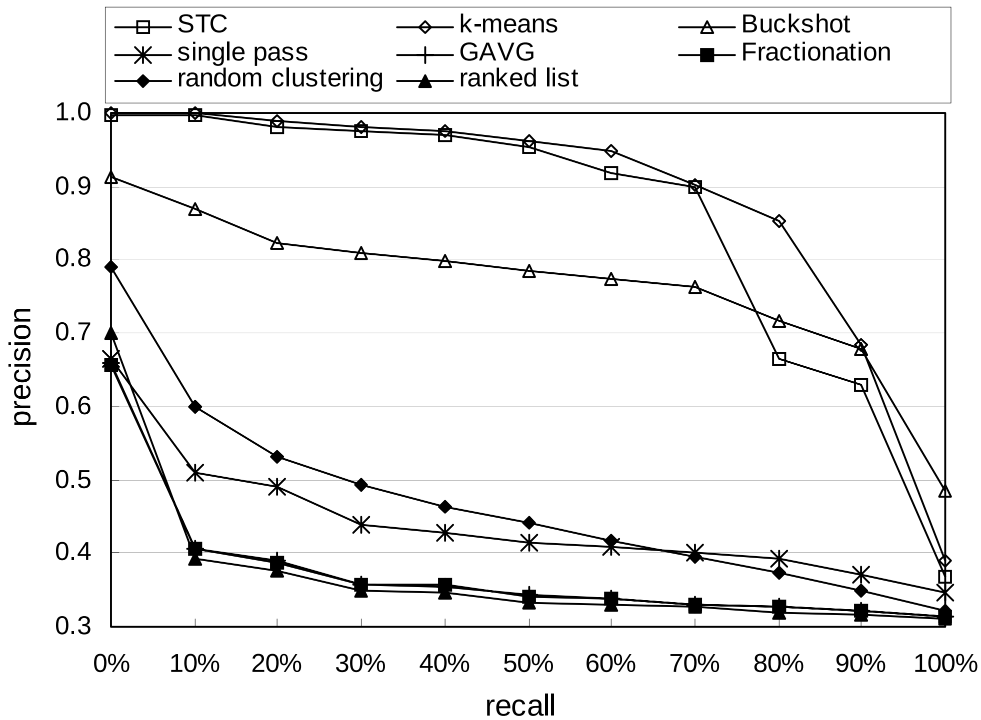


Figure 4.8: Precision-recall graph for the OHSUMED collection

The precision-recall graph for the results of the six clustering algorithms, the ranked list and random clustering for the OHSUMED collection. In this collection, we can see that k-means and STC perform quite similarly as the leading clustering algorithms, followed once again by Buckshot.

Figures 4.9, 4.10 and 4.11 present the Number-to-View graphs of the six algorithms, the ranked list and the random clustering. The Number-to-View graphs plot the Expected Search Length (the expected number of irrelevant documents that have to be viewed in order to encounter a certain number of relevant documents) as a function of the number of relevant documents the user wishes to view (1-10).

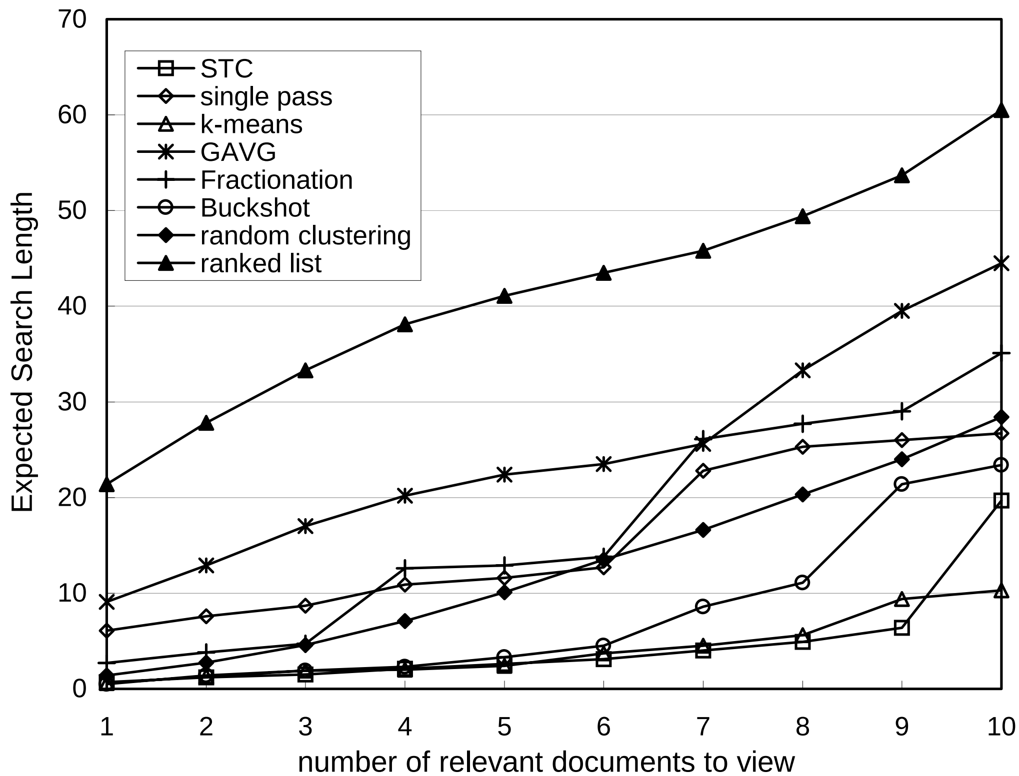


Figure 4.9: Number-to-View graphs for the WSR-SNIP collection

The Number-to-View graphs of the six clustering algorithms, the ranked list and the random clustering on the WSR-SNIP collection. This graph plots the Expected Search Length (ESL) as a function of the number of relevant documents the user wishes to view. STC and k-means are the two best performing algorithms

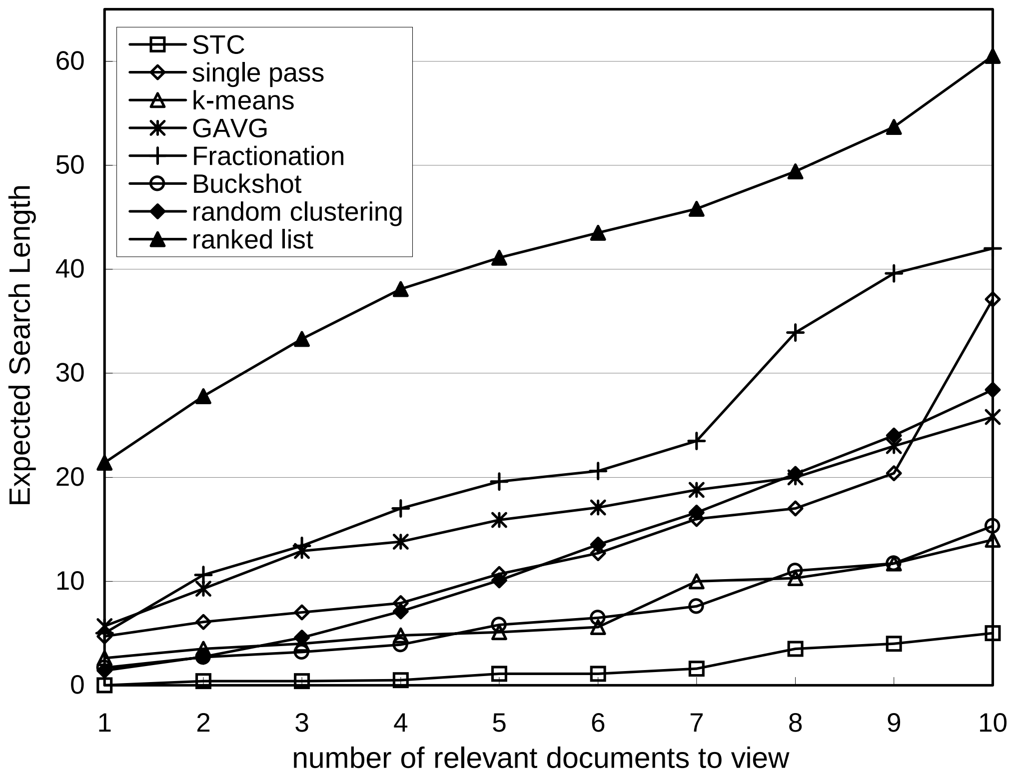


Figure 4.10: Number-to-View graphs for the WSR-DOCS collection

The Number-to-View graphs of the six clustering algorithms, the ranked list and the random clustering on the WSR-DOCS collection.

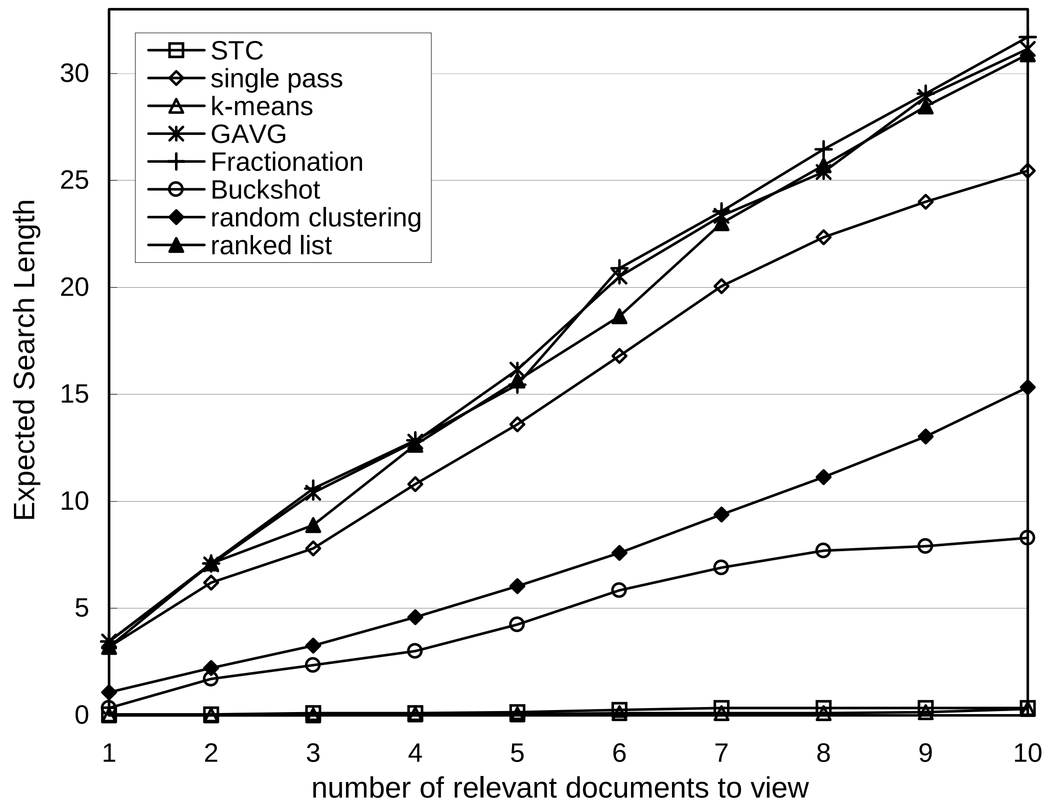


Figure 4.11: Number-to-View graphs for the WSR-OHSUMED collection

The Number-to-View graphs of the six clustering algorithms, the ranked list and the random clustering on the OHSUMED collection. Both k-means and STC appear to present to the user almost no irrelevant documents among the first 10 documents that are viewed (based on our user model of course).

From the experiments presented in this section, it can be concluded that STC, k-means and Buckshot (a variation of k-means) are the highest performing clustering algorithms. Of these, STC seems to perform the best on the WSR collection, while k-means is better on the OHSUMED collection.

It is interesting to note the failure of the GAVG algorithm as apparent from these experiments. We have attempted to investigate this "manually" by observing each merge step of the algorithm on selected document sets. We believe GAVG is extremely sensitive to its halting criterion, especially with the many outliers that are typical in these document collections. GAVG would often produce one very large cluster containing most of the documents, and several extremely small clusters that often represented “noise” (a small group of similar irrelevant documents). We have tried a modified halting criterion – halting the algorithms when the similarity of the two most similar clusters is less than a certain threshold – which resulted in improved performance. But we found that the performance varied significantly with this threshold, and for each document set the “optimal” threshold was quite different. Fractionation and the single pass algorithm probably suffered from similar reasons. Another increase in the performance of GAVG was achieved when we used yet another modification – we limited the size of any cluster to 10% of the document set size (a similar idea was suggested by Schütze – personal communication). In the experiments presented above (and in the rest of this work), we used a combined halting criteria of desired number of clusters and minimum similarity threshold (whichever came first). Additional halting criteria (some very complex) have been suggested in the literature [Milligan and Cooper, 85] but not implemented in this research, as GAVG is a quadratic time algorithm and therefore not a true candidate for our clustering task.

The performance difference between STC and other clustering algorithms on the WSR collections and the OHSUMED collection seemed to be consistent across different metrics. STC outperformed all other algorithms on the WSR collections, however on the OHSUMED collection k-means was the top performer. OHSUMED is a collection of medical abstracts in which the authors have attempted to convey as clearly as possible the content of the documents (OHSUMED document sets, thought half the length of WSR-DOCS document sets, contain more than twice the number of unique words – see section 4.7). We speculate that this causes the vector-representation of OHSUMED documents to be more informative and more accurate in representing the documents’ content than the vector-representation of Web documents. This in turn might improve the performance of vector-based algorithms such as k-means on the OHSUMED collection.

4.3.2 The "Merge-then-Cluster" Evaluation

Another common technique of evaluating clustering algorithms is to compare the results to the "true" cluster structure [Milligan et al. 83]. If this evaluation technique produces results that are similar to those obtained using the IR evaluation technique – this will greatly increase our confidence in the results.

Determining the "true" cluster structure of a given document set is, in practice, impossible. Not only would it take a huge amount of resources, but human indexers who were given such a task showed little agreement in their results, and actually rated each others' results rather poorly [Macskassy et al., 98]. Alas, IR methodology has shown that even boolean relevance judgements are controversial [Harter, 96]. The common approach is therefore to generate a synthetic data set where the "true" clusters are known. This can be done by generating it based on a given cluster model, or, and more suitable for document clustering, by merging distinct sets of documents into one. The resulting document set is then clustered using different clustering algorithms, and the results are then evaluated given how closely they correspond to the original partition. We call this the "merge-then-cluster" evaluation approach.

4.3.2.1 Methology

In section 4.2 we described how we created 60 OHSUMED document sets by merging 2, 4 or 8 distinct document groups (groups of document relating to a single topic in the MeSH subtree of "Cardiovascular"). Following we shall describe the different metrics we used to evaluate how well the generated clusters correspond to the original partition of the documents into groups:

Precision Factor: This metric is commonly used in the Topics Detection and Tracking (TDT) domain. For each cluster the algorithm identifies, we determine what is the most frequent topic in the cluster and then calculate that cluster's precision as the number of documents relating to the most frequent topic divided by the total number of documents in the cluster. We define the *precision factor* as the weighted average (by cluster size) of the precision of all the clusters. But what should we do regarding all the unclustered documents (documents for which no significant similarity to other documents was found)? In clustering algorithms there is often a tradeoff between the quality of the clustering and the size of the unclustered group. Therefore we also introduce the *normalized* precision factor which is equal to the precision factor multiplied by the fraction of the documents that were actually clustered.

Pair-wise Accuracy: We start by looking at all pairs of documents within each cluster, and count the number of true-positive pairs (the two documents were originally in the same group) and false-positive pairs (they were not originally in the same group). Our pair-wise accuracy metric is a normalized measure of how many more true-positive pairs we have as opposed to false-positive pairs. These types of metrics are commonly used in the field of Cluster Analysis. Let C be a set of clusters, $tp(c)$ and $fp(c)$ be the number of true-positive pairs of documents and the number of false-positive pairs in cluster c of C. Let $uncl(C)$ be the number of unclustered documents in C. We define the pair-wise score of C, PS(C), as:

$$\mathrm{PS}(C) = \Sigma\ \mathrm{sqrt}(tp(c)) - \Sigma\ \mathrm{sqrt}(fp(c)) - uncl(C)$$

where the summations are over all clusters c in C. We use the square roots of $tp(c)$ and $fp(c)$ to avoid over-emphasizing larger clusters, as the number of document pairs in a cluster c is $|c| \cdot (|c|-1) / 2$. We subtract the number of unclustered documents, as these documents are misplaced. The maximum pair-wise score of C, maxPS(C), is:

$$\mathrm{maxPS}(C) = \Sigma\ \mathrm{sqrt}(|c| \cdot (|c|-1) / 2)$$

where $|c|$ is the number of documents in cluster c. This is simply the sum of the square roots of the number of document pairs in each cluster. Finally, we define the pair-wise accuracy quality measure of C, PA(C) as:

$$\mathrm{PA}(C) = (\mathrm{PS}(C) / \mathrm{maxPS}(C) + 1) / 2$$

Which is pair-wise score divided by the maximal score possible and normalize to the range of 0 to 1. As before we also introduce the *normalized* pair-wise accuracy metric, which is equal to the pair-wise accuracy multiplied by the fraction of the documents that were actually clustered.

4.3.2.2 Results

Figures 4.12 and 4.13 compare the precision factor and the pair-wise accuracy (respectively) of the six clustering algorithms and of random clustering on the OHSUMED text collection. It is encouraging that the results using both metrics are quite similar. From Figure 4.12 we see that STC achieves the highest precision factor, slightly ahead of k-mean and Buckshot (statistically significant: $p = 0.41$). But STC has a larger unclustered group, and therefore its normalized precision factor is slightly less than these two algorithms (statistically significant difference to k-means: $p = 0.004$; no significant difference to Buckshot). Fractionation, GAVG and single pass appear to do poorly by this metric. Using the pair-wise accuracy metric (Figure 4.13) k-means appears to be performing slightly better than STC using the normalized metric (no statistical difference for the non-normalized metric between k-means and STC, nor between STC and Buckshot), with the rest of the results unchanged.

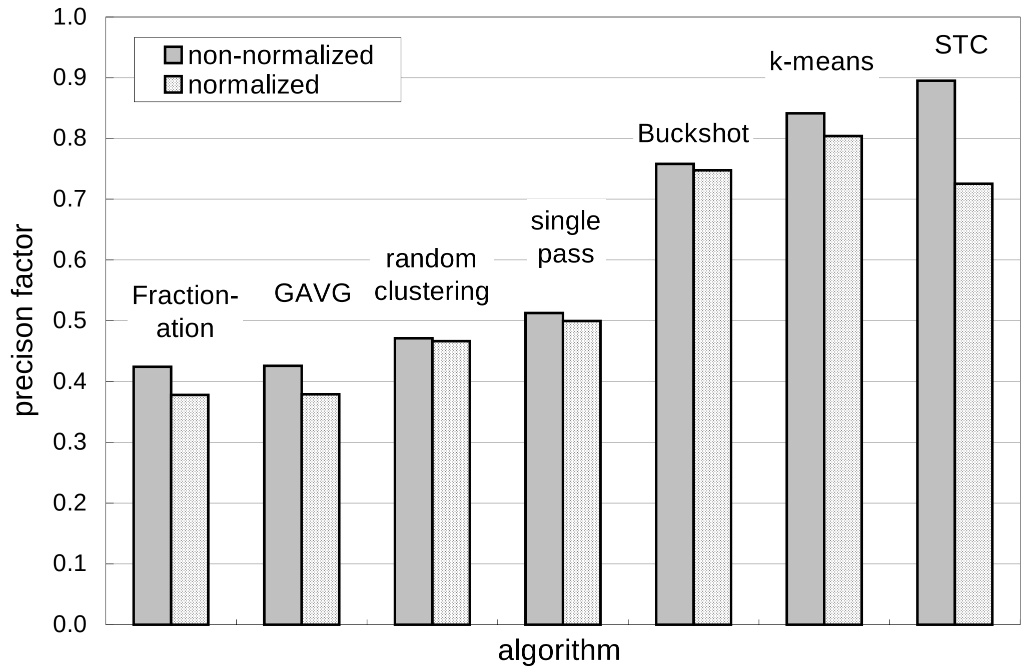


Figure 4.12: Precision factor

The precision factor of the six clustering algorithms and of random clustering on the OHSUMED collection.

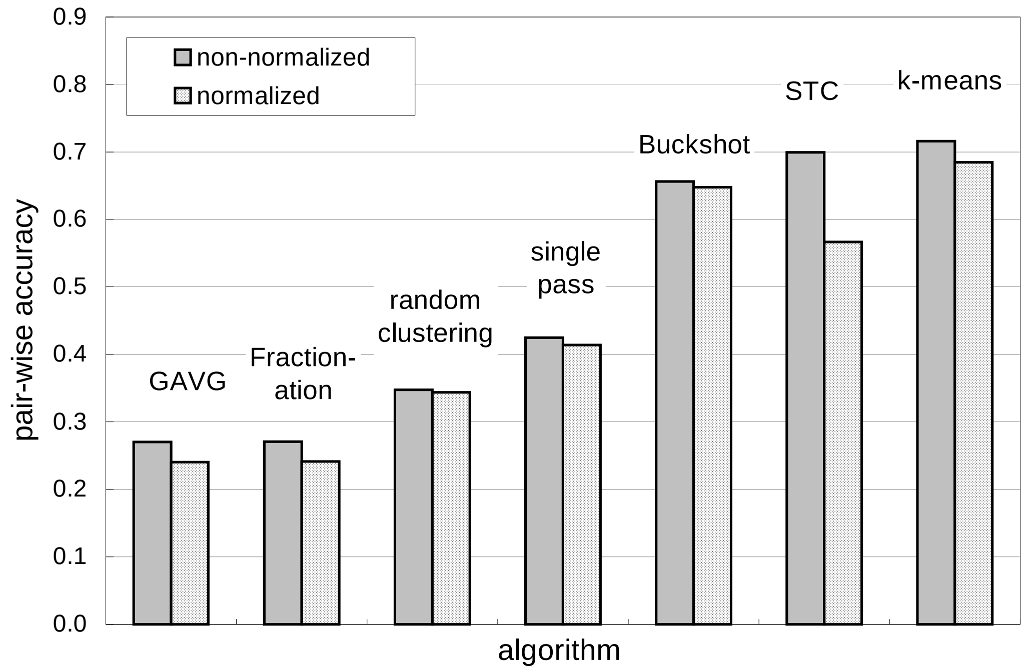


Figure 4.13: Pair-wise accuracy

The pair-wise accuracy of the six clustering algorithms and of random clustering on the OHSUMED collection.

4.3.3 Summary

In this section, we evaluated the “clustering quality” of STC as compared five other clustering algorithms on three text collections. The clustering quality was compared using two quality-evaluation approaches: the IR approach (for all three text collections) and the “merge-then-cluster” approach (for the OHSUMED collection).

We tried to use a large number of varied methods to evaluate the clustering quality as each method can easily be criticized. The IR approaches impose an unrealistic user model and then apply (controversial) metrics developed for a different task. The “merge-then-cluster” approach creates synthetic document sets and uses metrics that are not task driven. Nonetheless, the consistency of the results across the different evaluation metrics and text collections is encouraging and allows us to draw cautious conclusions.

We can conclude that STC, k-means and Buckshot are the highest performing clustering algorithms on our text collection. Of these, STC seems to perform the best on the WSR collections, while k-means was slightly superior on the OHSUMED collection. Buckshot appears to consistently come in third place. The other three clustering algorithms – Fractionation, GAVG and single pass performed rather poorly by our evaluation metrics. Earlier in the section, we have hypothesized several reasons for this poor performance including sensitivity to parameter settings (halting criteria and thresholds) and poor performance with many outliers.

In the next section we shall investigate the contribution of two of STC’s characteristics to its success: its use of phrases and its overlapping clusters.

4.4 Contribution of Phrases and of Overlapping Clusters

We believe that the positive results achieved by the STC algorithm were due in part to the fact that STC uses phrases and to the fact that it naturally allows overlapping clusters. In our experiments on the WSR-DOCS collection, 55% of the phrase clusters (in the top 10 clusters) were based on phrases containing more than one word. STC placed each document in 2.1 clusters on average and 72% of the documents were placed in more than one cluster.

To measure the impact of these features on STC's performance, we ran an ablation study in which we created two hobbled variants of the STC algorithm. In the first variant – *STC-no-overlap* – we ran a post-processing phase that eliminated any overlap between the clusters by removing each document that was placed in several clusters from all but one cluster, the one whose centroid was the closest to the document. In the second variant – *STC-no-phrases* – we allowed STC to use only single word phrases.

Figure 4.14 presents the performance of these variants as compared to the original STC algorithm. The figure presents the average precision metric (using the 10% viewed user model) on the WSR-DOCS collection. In this case we treat documents viewed more than once as irrelevant (*e.g.* the "STC–min" metric) as this will allow a more objective comparison to the non-overlapping version of the algorithm.

We see that both cluster overlap and multiword phrases are critical to STC's success (p = 0.039 significance difference between STC and STC-*no-phrases*; p = 0.07 between STC and STC-*no-overlap*). Phrases are key because they are the basis for identifying cohesive clusters; overlap is key because we have no means of deciding which phrase in a document ought to determine its assignment to a cluster. Overlap enables the document to potentially participate in all clusters that are based on shared phrases from that document.

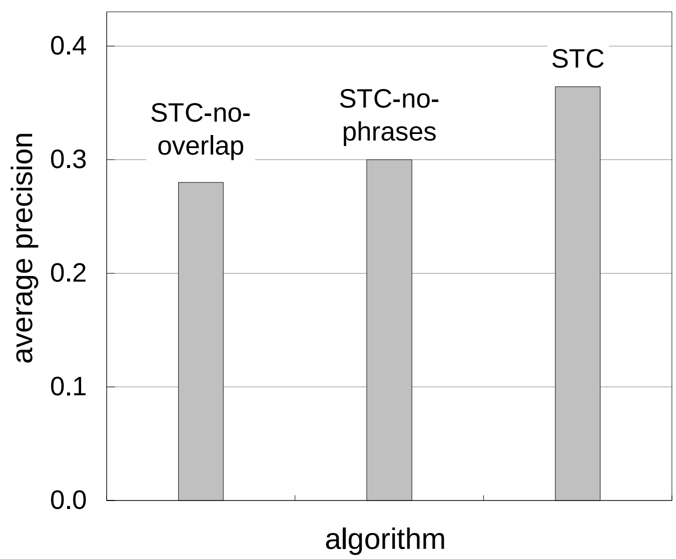


Figure 4.14: Average precision of STC variants

The average precision (using the 10% viewed user model) on the WSR-DOCS collection of the variants of STC: *STC-no-overlap* which forces the results into a true partition and *STC-no-phrases* which uses only single word phrases.

The intuition for why overlapping clusters can improve the precision of a clustering algorithm is as follows. The non-overlapping version of a clustering algorithm might place a relevant document in a cluster that is not the "best" cluster (the first one the user will view according to our model – the one with the highest density of relevant documents). The overlap-producing version might also place this document in the "best" cluster as well. This will happened if a relevant document is similar to the "best" cluster, but more similar to another cluster, as might happen if, for example, the document relates to multiple topics or shares common features with irrelevant documents (*e.g.*, same source or author). On the other hand overlapping clusters might cause the same document to be viewed more than once, which will decrease the performance of the algorithm by our the evaluation metrics.

We also examined the effect of allowing overlapping clusters on the additional cluster algorithms. We modified Buckshot and k-means to allow overlapping clusters by allowing each document to be placed in more than one cluster in the last iteration of the algorithm. We chose these two algorithms for this comparison, as they can easily be adapted to produce overlapping clusters. The results are presented in Figure 4.15. We can see that allowing overlapping clusters slightly improves the average precision of both Buckshot and k-means (not statistically significance).

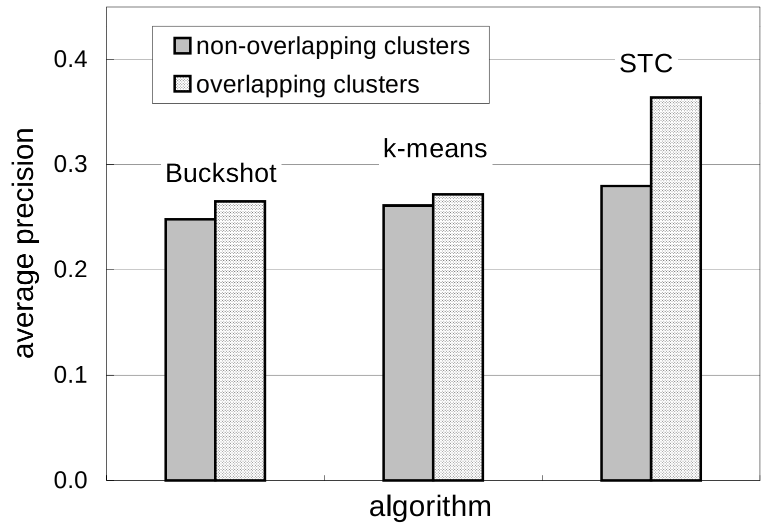


Figure 4.15: Effects of overlapping clusters

The average precision of overlap producing clustering algorithms compared to their non-overlapping versions on the WSR-DOCS collection. Allowing overlapping clusters improves the average precision of both Buckshot and k-means.

4.5 Contribution of Phrases

One of STC's main features is its use of phrases, while the other vector-based algorithms that were compared to it use single words. 55% of the phrase clusters (in the top 10 clusters) were based on phrases containing more than one word. Can other clustering algorithms also benefit from using phrases? In this section we investigate this question.

To measure the impact of phrases on other clustering algorithms, we compared the standard algorithms which use only single words as document vector attributes, to modified versions which also use multi-word phrases (identified via a suffix tree) as attributes. The procedure was as follows. A suffix tree was constructed from the document set and its nodes were scored as in the STC algorithm; the top 500 phrase clusters were collected. The phrases of these phrase clusters that contained more than one word were added as attributes to the vectors of all the documents containing them.

Figure 4.16 presents the average precision of STC and the five vector-based clustering algorithms with and without multiword phrases on the WSR-DOCS collection. The results show that using phrases increases the performance across all algorithms ($p = 0.039$ for STC, single pass and GAVG; $p = 0.004$ for Fractionation; difference not statistically significant for k-means and Buckshot). It is most dramatically apparent for the GAVG, Fractionation and single pass algorithms, which performed poorly when using only single words, but performed comparably to the other algorithms when using phrases. Previously, we hypothesized that their poor performance was the result of the “noisiness” (*i.e.*, many outliers) of the document sets and their sensitivity to the halting criteria. We believe that using phrases reduces the amount of noise in the data, as chance statistical similarity is less likely to occur with phrases. This is similar to the observed phenomenon in IR that phrases typically increase precision at the expense of decreasing recall [Furnkranz et al., 98].

We ran the same experiment on all three document collections. The results across all three collections showed that using phrases increase the average precision, but the degree of increase was quite different among the different collections. In the WSR-SNIP collection the improvement was modest (14% when averaged across all algorithms), and actually was not apparent for all algorithms. In the WSR-DOCS collection the improvement was more apparent (52%), and in the OHSUMED collection it was the highest (79%). The reason for this is possibly the importance of phrases in defining topics in each collection. OHSUMED is a scientific medical collection and it is probable that common phrases often recur in documents that relate to the same topic. In section 4.7, we presented the average number of phrases identified in document sets from each collection and showed it to be highest for OHSUMED, a fact that can also help to explain these results.

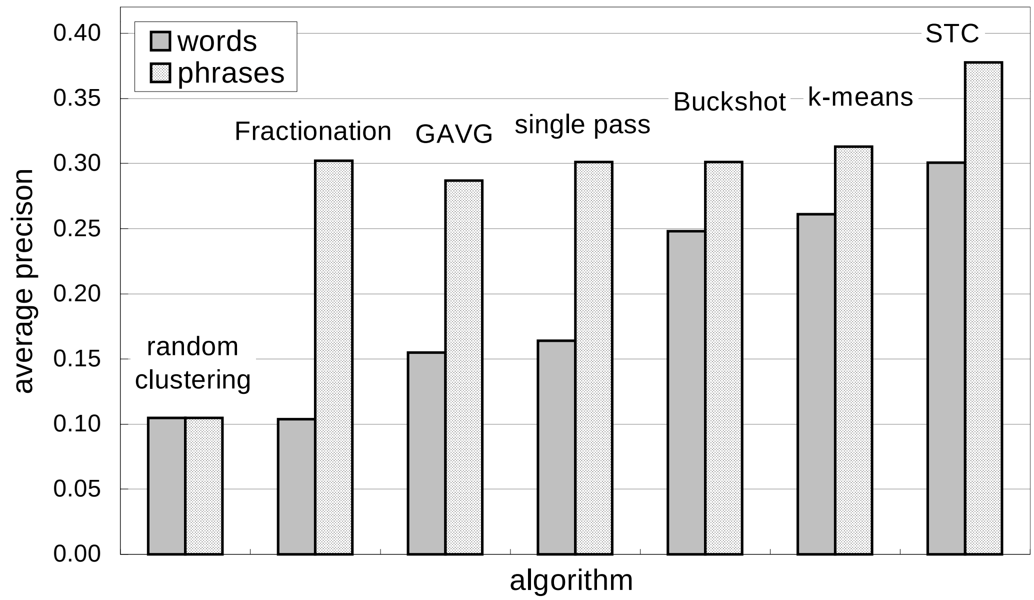


Figure 4.16: Contribution of using phrases in clustering

The average precision of the six clustering algorithms with and without multiword phrases on the WSR-DOCS collection. We created a hobbled variant of the STC algorithm that uses only single word phrases. For the other clustering algorithms, we compared the standard algorithms, which use only single words as document vector attributes, to modified versions which also uses multiword phrases (identified via a suffix tree) as attributes.

Can we attribute these results to the fact that we are simply using more attributes? In these experiments we have increased the average length of a document's vector from 87 to 124 attributes for WSR-DOCS document, from 8 to 12 for WSR-SNIP documents, and from 37 to 57 for OHSUMED documents. It appears that across all three collections approximately 50% more attributes were added to each vector. It is important to note that the total increase in the number of attributes across all documents was much less (as we look at only the top 500 base clusters) – between 10% (OHSUMED) and 20% (WSR-SNIP). The improvement of the clustering algorithms on the WSR-SNIP collection was relatively small compared to the other collections, even though the lengths of the vectors were increased by 50% and the number of total attributes was increased by 20%. We therefore conclude that the shown improvements do not result merely from adding additional attributes. We speculate that the clustering quality has improved as phrases provide additional information relating to the co-occurrence of words.

4.6 Comparing Different Phrase Generation Techniques

We have shown that phrases are key to STC's ability to create high quality clusters and that they can increase the performance of other clustering algorithm. In this section we are investigate what makes phrases useful for clustering. First we compare the contribution of phrases of different size – are longer phrases better than shorter ones? In section 4.6.1 we compare the performance of STC using suffix tree phrases and using n-grams: phrases created by using sliding windows of *n* words over the text (typically 2-grams are used, occasionally also 3-grams) [Brown et al., 90].

We also wish to understand why phrases are better than words: Does the advantage of using phrases lie in the information present in the adjacency and order of the words, or simply in their being multiword features? And if the latter is the case, can we use other multiword features with the same success? To address these issues we generalized the STC algorithm such that phrase clusters are defined using frequent sets – sets of documents that share a set of words. We call this version of the STC algorithm Frequent Set Clustering (FSC). In section 4.6.2 we compare the word-based and the phrase-based variants of STC in terms of speed and of clustering quality.

4.6.1 Suffix Tree Phrases vs. N-Grams

A suffix tree has two advantages as the mechanism for phrase identification. First, it can find phrases of any length, while the alternative techniques all look at relatively short phrases. Second, it is fast and efficient in finding phrases shared by two or more documents. It is efficient in the sense that most nodes in the suffix tree correspond to maximal phrase clusters. However, both lexical affinity phrases and n-grams can be implemented just as fast.

Before comparing n-grams and suffix tree phrases it is interesting to note the characteristics of the phrases identified by the suffix tree. In Table 4.1 we present the distribution of the lengths (number of words) of the phrases that were identified by the suffix tree, and of the phrases that were used as features in the documents' vector representation. Identified phrases include all phrases that occur in a minimum number of documents. Of these, the algorithm selects the 500 highest scoring phrases (using the same scoring function as in the STC algorithm) to be considered as features in the document's vector representation. From the table, the dominant importance of phrases of length 2 (2-grams) can be seen. Nearly 50% of the phrases used are of length 2, even though they comprise less than 30% of the phrases identified.

Table 4.1: Lengths of phrases identified and used by STC

Num. Of words in	num. of phrase identified	num. of phrases used
2	9230.3	205.4
3	6369.4	84.3
4	4379.6	46.9
5	2999.4	33.9
6	2075.6	22.1
7	1429.6	14.0
8 or more	3378.9	21.6

The distribution of the lengths of the phrases identified by the suffix tree, and of the phrases used as features in the documents' vector representation. Identified phrases include all phrases that occur in a minimum number of documents. Phrases used are the 500 highest scoring phrases (using the same scoring function as in the STC algorithm). Nearly 50% of the phrases used were of length 2.

In Figure 4.17 we compare the clustering quality of STC using phrases identified via a suffix tree as compared to n-grams. In one instance we use single words and 2-grams as our phrases; in another we also use 3-grams. This experiment was run on all three document collections. It is surprising to note that there was very little difference (no statistically significant difference) between the performance of suffix tree phrases and of n-grams. The average difference in performance using the two kinds of phrases was less then 4% for this collection (it was even smaller – less than 2% – for the OHSUMED collection). This is consistent with unpublished work showing little difference between the use of 2-grams and of suffix tree phrases in document classification [Dumais and Zamir, 99].

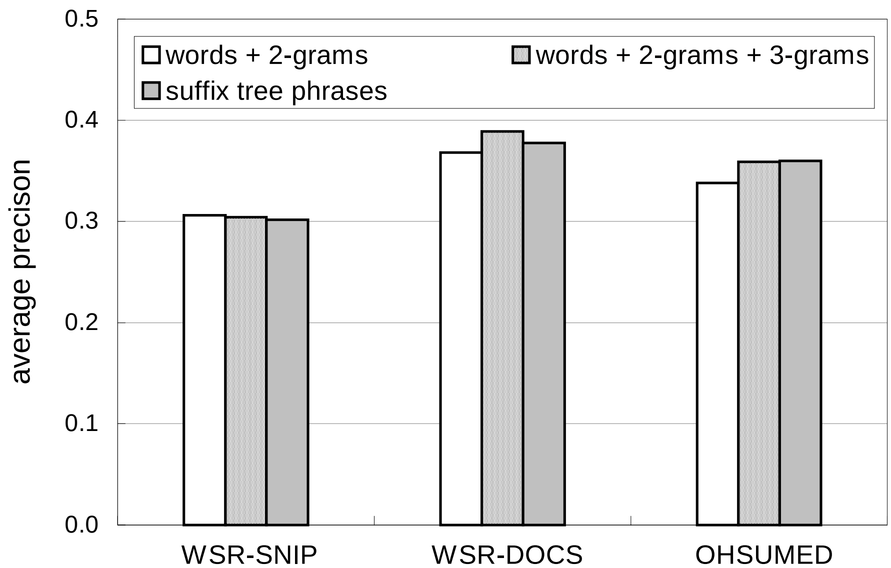


Figure 4.17: Suffix tree phrases vs. n-grams

The average precision of the six algorithms using either single words, phrases identified via a suffix tree or 2-gram phrases on the WSR-DOCS collection. From this experiment, there appears to be little difference between the performance of suffix tree phrases and of 2-grams.

We conclude that regarding clustering quality, suffix tree phrases show little advantage over other kinds (perhaps simpler) of phrase generation techniques, such as 2-grams. This is consistent with previous work that showed document classification improved when 2-grams were added (a slight improvement was also achieved by adding 3-grams), while longer n-grams actually decreased performance [Furnkranz, 98; Mladenic and Grobelnik, 98]. We believe that the main advantage of using suffix tree phrases lies in their descriptive power – longer phrases are much better at describing the contents of a cluster to the user.

4.6.2 Word-Set Based Clustering

Why are phrases better than words? Is it simply because they are multiword features or is the information present in the adjacency and order of the words important? To address this issue we generalized the STC algorithm such that phrase clusters are defined using non-sequential multiword features.

The STC algorithm creates clusters using phrase clusters, which are sets of documents that share a common phrase. However, we can generalize the STC algorithm such that phrase clusters are defined as sets of documents that share a common feature, not necessarily a phrase. For example, a feature can be a lexical affinity pair (a pair of words occurring within 5 words of each other) [Maarek and Wecker, 94], or an approximate phrase with some mismatches. Another option, which we shall investigate in this section, is to use *maximal frequent sets* as the features that define phrase clusters. We call this version of the STC algorithm Frequent Set Clustering (FSC). We shall first define maximal frequent sets, and discuss methods of discovering them efficiently. Next, we shall compare the word-based and the phrase-based variants of STC in terms of speed and of clustering quality.

4.6.2.1 Frequent Sets

Frequent sets have been recently studied in the field of Data Mining. They have been used for the task of association-rule discovery [Agrawal et al., 93], and to find correlation between terms in a database [Feldman et al., 97]. We shall first define frequent sets and then describe generating algorithms.

> **Definition:** Given a set of documents *D*, a *frequent set f* in *D* is a set of words that occur together in at least *minsup* documents in *D*. The number of documents that *f* occurs in is called the support of *f*. A *maximal* frequent set is a frequent set that has no supersets that is frequent.

Several algorithms have been proposed for the generation of frequent sets. Almost every recently proposed algorithm is a variant of Apriori [Agrawal and Srikant, 94]. Apriori employs a bottom-up search that enumerates all frequent sets. Therefore, it is inherently exponential in the length of the longest frequent set, as a frequent set of length l has 2^l subsets, which must be frequent as well. The methods used to limit the execution time of such algorithms is by increasing the *minsup* required to support a frequent set, and by aggressive feature selection to reduce the number of feature in each transaction (a transaction is equivalent to a document in consisting of a set of tokens).

The typical data-mining scenario, for which these algorithms were developed, is the discovery of frequent sets in a very large database of short transactions. We, however, wish to discover frequent sets in a small set (several thousands) of very long transactions (documents) over a large vocabulary (all possible words). Recent work has demonstrated that Apriori-like algorithm are inadequate for data sets that contain large frequent sets [Brin et al., 97; Bayardo, 97]. Several enhancements were proposed to help address these limitations by searching only for maximal frequent sets [Zaki et al. 97; Lin and Kedem, 98; Bayardo, 98]. These improvements are still exponential in the worse case analysis, but have been shown to perform considerably better in some domains.

4.6.2.2 Quality Evaluation

The FSC algorithm uses phrase clusters defined by maximal frequent sets – frequent sets that have no supersets that are frequent. This helps the algorithm prune the search space but it is also important in terms of clustering quality. The reason for this is that nonmaximal frequent sets are usually redundant in the information they represent, but they "divert" resources and attention from other non-redundant information. To illustrate this, Table 4.2 displays the number of frequent sets and maximal frequent sets discovered for a specific WSR-DOCS document set of 200 documents. We can see that although there are only 4 maximal frequent sets of length 9 or more, there are 1394 frequent sets those lengths. Thus, when FSC selects the 500 highest scoring phrase clusters, they probably all relate to a small number of documents.

Table 4.2: Numbers of frequent sets and of maximal frequent sets

frequent set size:	**1**	**2**	**3**	**4**	**5**	**6**	**7**
num frequent sets:	843	865	1405	2194	2829	3017	2627
num maximal frequent sets:	76	199	58	26	19	12	17

frequent set size:	**8**	**9**	**10**	**11**	**12**	**13**	**total**
num frequent sets:	1804	937	352	90	14	1	14149
num maximal frequent sets:	4	2	0	0	1	1	415

The number of frequent sets and maximal frequent sets discovered for a specific WSR-DOCS document set of 200 documents. We can see that although there are only 4 maximal frequent sets of length 9 or more, there are 1394 frequent sets those lengths.

The clustering quality of the FSC algorithm is presented in Figure 4.18. The Figure compares the average precision of FSC and STC, given the 10%-viewed user model, on the three document collections. FSC surpasses STC on the short documents of the WSR-SNIP collection (not statistically significant). STC performs slightly better on the WSR-DOCS collection (p = 0.065) and significantly better on the OHSUMED collection (p = 0.004). This might indicate that in the WSR-SNIP collection phrases add relatively less information that in the other collections.

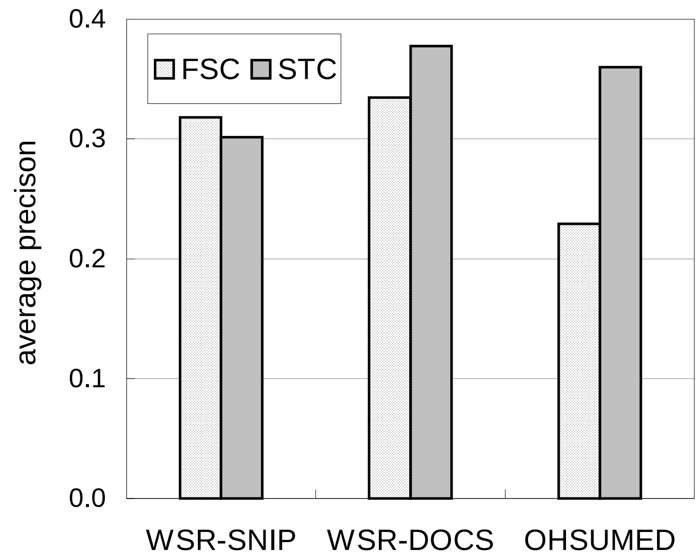


Figure 4.18: Average precision of FSC compared to STC

The average precision of FSC and STC, given the 10%-viewed user model, on the WSR-SNIP, WSR-DOCS and OHSUMED collections. FSC surpasses STC on WSR-SNIP collection, but is inferior to STC on the other two collections. This might indicate that in the WSR-SNIP collection phrases add relatively less information that in the other collections

4.6.2.3 Speed

In order to achieve fast frequent sets discovery time, which is essential for the FSC clustering algorithm, we rely on four implementation design choices:

1. We search only for maximal frequent sets, not for all frequent sets, and we use a pruning technique which is very similar to the one used by Bayardo [Bayardo, 98]. This pruning allows the algorithm to abandon the strict bottom-up search, and instead attempts to "look ahead" in order to find long frequent sets early on. This, in turn, allows the algorithm, under certain condition, to prune subsets of these long frequent sets, and thus avoid the need to enumerate all frequent sets. The use of maximal

frequent sets is also extremely important for the quality of the FSC algorithm, as will be elaborated on in the next section.

2. For our document clustering task, the frequent sets algorithm has to handle a relatively small number of long transaction (several hundreds or thousands documents) over a large vocabulary (the English language). This leads to the observation that the frequent sets in this domain typically have a small number of documents associated with them. We therefore modify our basic Apriori-style algorithm by associating with each frequent sets a sorted list of the ids of the documents that contain it. Therefore when we create a new frequent set by merging two smaller ones, its support can be obtained by performing an intersection of their two document lists. This would not be feasible under the standard domain assumptions of discovering frequent sets in large databases, as then each frequent set might be associated with millions of documents. This approach also allows us to avoid enumerating all frequent sets as, for example, we can find all frequent sets of length 8 from frequent sets of length 6, without enumerating all frequent sets of length 7.
3. We use an adaptive *minsup* support threshold. We start with a 2% threshold, but change it to 3% and then to 4% if we discover too many frequent sets (more than 10000) in any step of the algorithm.
4. When analyzing Web search engine results, large frequent sets are often found when a certain document appears several times in slightly different variations. If the document is of considerable length this will lead to a very expensive computation. We therefore employ a “near duplicate elimination” scheme in which we try to identify documents which are highly similar, then remove one of them from consideration and simply add it to the cluster that its “duplicate” ended up in. This can be done before the frequent set discovery phase, but that would be quadratic in the number of documents. We employ an alternatively technique which turns out to be much faster in practice. When too many (over 10000) frequent sets are discovered in a certain step of the algorithm (at a 4% support level), we perform the following duplicate

identification procedure. We make a pass through all the frequent sets and calculate for each document pair its *frequent sets intersection ratio* – the number of frequent sets that the two documents appear in together out of the total number of frequent sets that either one of then appears in. If this ratio is large, we mark these two documents as duplicate and remove one of them from consideration. This process is linear in the number of frequent sets at that stage, but quadratic in the length of the longest document list of a frequent set. This is an expensive process, but it is much faster in practice, for large document sets, than finding near-duplicates by comparing all pairs of documents to each other.

In the next section we compare the speed of the STC algorithm to other clustering algorithms. The execution time of the FSC algorithm is also presented (Figure 4.19), and is comparable to that of STC. On the WSR-SNIP collection, FSC is some 25% slower than STC (clustering 500 documents), on the WSR-SNIP collection it is 20% faster and on the OHSUMED collection it is 45% slower. The speed of FSC is highly dependent (and exponentially so in the worse case) on the length of the largest frequent sets in the data. Therefore it is not surprising that its performance is extremely collection sensitive.

4.7 Speed

In this section we describe experiments performed to compare the speed of the various clustering algorithm. Seven clustering algorithms were compared: Buckshot, Fractionation, group-average hierarchical clustering (GAVG), k-means, single-pass, STC and FSC. The GAVG algorithm was chosen as it is commonly used; the rest were chosen as they are fast enough to be contenders for online clustering. We measured the execution time of these algorithms while clustering 10 document sets of variable sizes (100 to 1000 documents). We ran the algorithms with parameters that have been shown to produce good results (in the cluster-quality experiments described in the next section), and all were optimized to the same degree.

Some implementation details are needed: Buckshot and k-means were run with 3 assign-to-nearest iterations. In all the vector-based algorithms (everything besides STC), the word vectors were truncated to the 64 most significant words to increase speed (this was shown to cause little degradation in the quality of the clustering [Schütze and Silverstein, 97]).

In Figures 4.19, 4.20 and 4.21 we present the time to cluster WSR-SNIP, WSR-DOCS and OHSUMED document sets respectively. Each reported time is averaged over 10 document sets of the same size. The times were measured using a Linux machine running on a Pentium 200 processor with 64MB of RAM. Using a stronger machine we can expect to reduce the times shown by a factor of two. We also know how to improve the efficiency of our code to obtain another estimated 2-fold speedup.

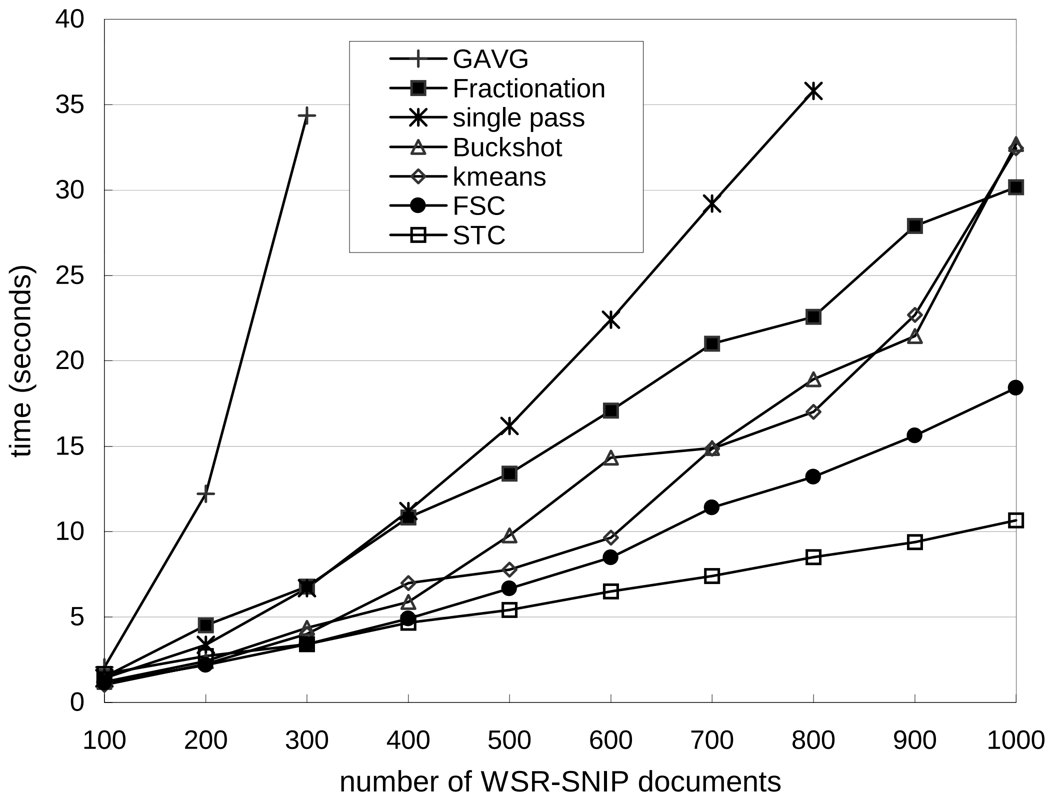


Figure 4.19: Execution time on WSR-SNIP documents

Execution time (in seconds) of the different clustering algorithms on WSR-SNIP document sets as a function of the set size. STC is the fastest algorithm on the short Web snippets, clustering 1000 snippets in 11 seconds. By improving the code and using a faster machine we estimate we can achieve a 4-fold speedup in the execution time.

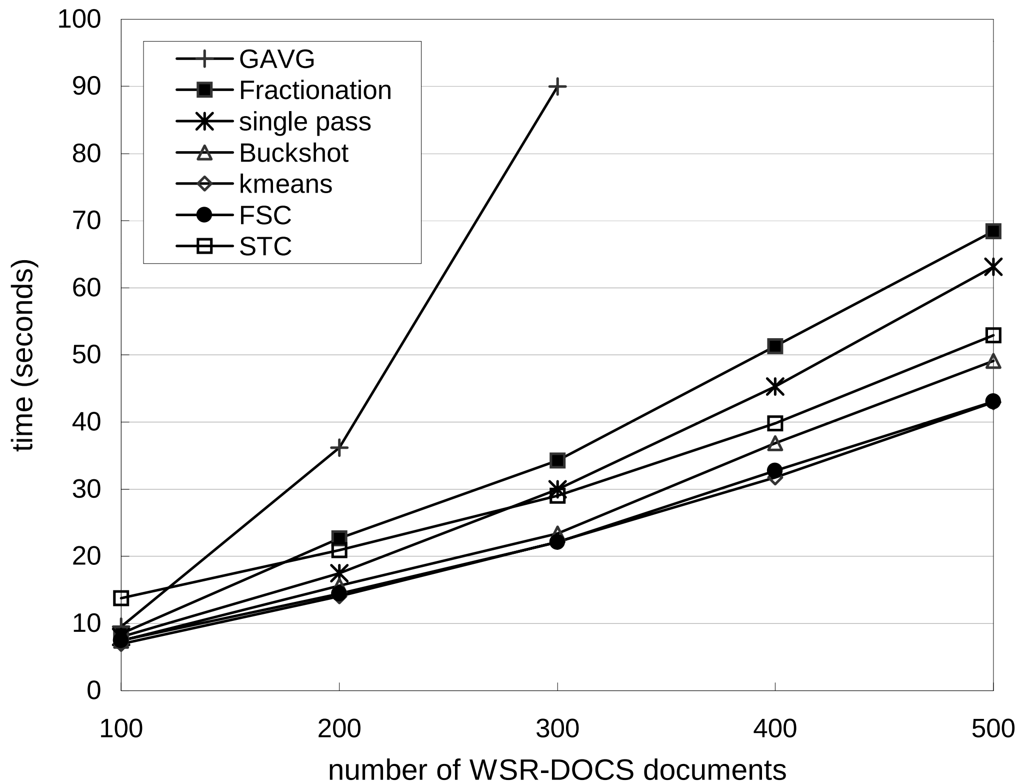


Figure 4.20: Execution time on WSR-DOCS documents

Execution time (in seconds) of the different clustering algorithms on WSR-DOCS document sets as a function of the set size.

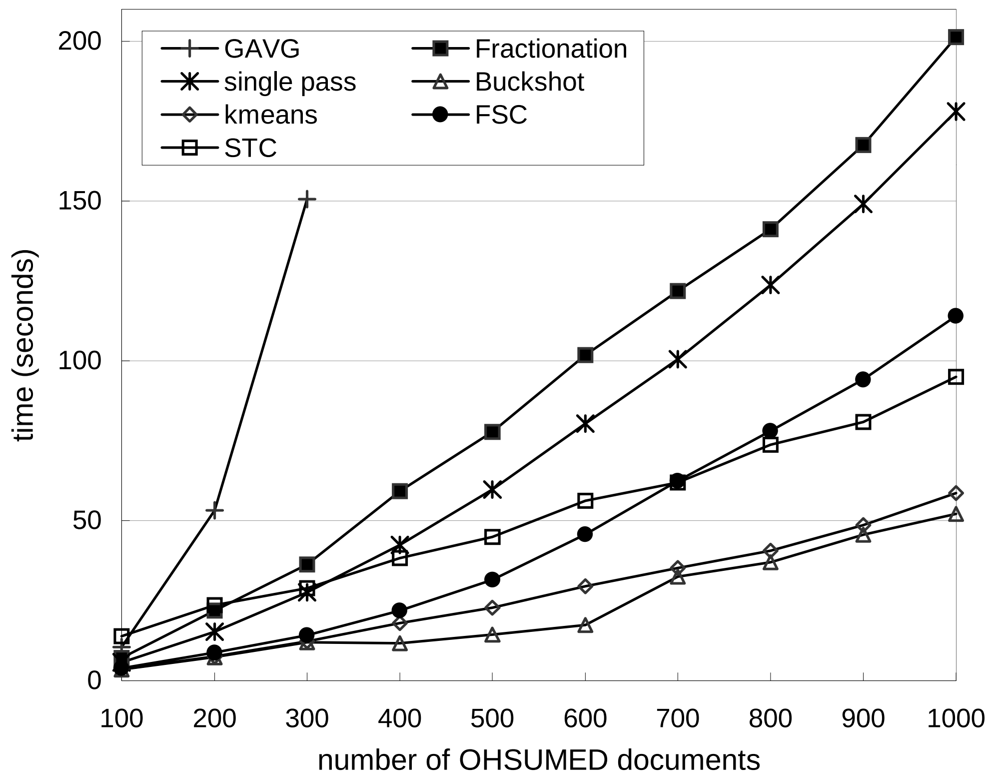


Figure 4.21: Execution time on OHSUMED documents

Execution time (in seconds) of the different clustering algorithms on OHSUMED document sets.

It is plain from the poor performance of the group-average hierarchical clustering algorithm that only near linear time algorithms can produce clusters for sets of hundreds of documents fast enough for true online interaction. On all document collections Buckshot and k-means appear to perform similarly (as can be expected as Buckshot is a variation on the k-means algorithm), and outperform Fractionation and single-pass. STC shows better performance than Buckshot and k-means on the WSR-SNIP collection, and slightly worse performance on the OHSUMED collection.

The comparison above is conservative as is does not take into account the incremental nature of STC. As argued in the introduction, our model for document clustering is that the clustering should be done on a machine separate from the search engine server. This “clustering server” (which can be the users client PC) receives search engine results over the Web and output clusters to the user. Because STC is incremental, it can use the "free" CPU time in which the system is waiting for the search engine results to arrive over the Web. Therefore, if the Web delay is substantial (on the order of the clustering time), STC would produce results instantaneously after the last document arrives, while the non-incremental algorithms will only start their computations. Being incremental also enables the system to instantaneously display results when an impatient user interrupts the clustering algorithm, and allows it to be used for event detection and tracking tasks.

We can attempt to explain the different performance of STC and the k-means algorithms on the different collections by investigating some characteristics of these collections. Table 4.3 presents the average document length in the three collections, with and without stopwords. While all the algorithms compared (asides from the hierarchical algorithm) are linear both in the number of documents and in the average length of a document, the constants are different. The vector-based algorithms are linear with the average length of a document because the documents have to be parsed to create the word vectors. Once the word vectors are created, the algorithms progress using constant length vectors, thus are not dependent any more on the documents’ length. On the other hand, STC depends on the average document’s length both when parsing the document and when building the suffix tree, thus the document-length constant is obviously larger. This helps explain why STC is faster than k-means for example on the WSR-SNIP collection, but is slightly slower on the WSR-DOCS collection.

Table 4.3: Average number of words per document

	number of words	number of non-stopped words
WSR-SNIP	40	39
WSR-DOCS	476	388
OHSUMED	206	199

The average number of words per document in the three document collections. The second column represents the number of words that do not appear in the English stoplist.

Another observable difference between the document collections is the “richness” in their vocabulary. Figure 4.22 present the average number of unique “non-trivial” words in a document set (words that are not stopped and that appear in at least a minimal number of documents in the set) as a function of the document set’s size for the three document collections. We can see that although OHSUMED documents are, on average, shorter than WSR-DOCS documents, they contain a larger vocabulary of words.

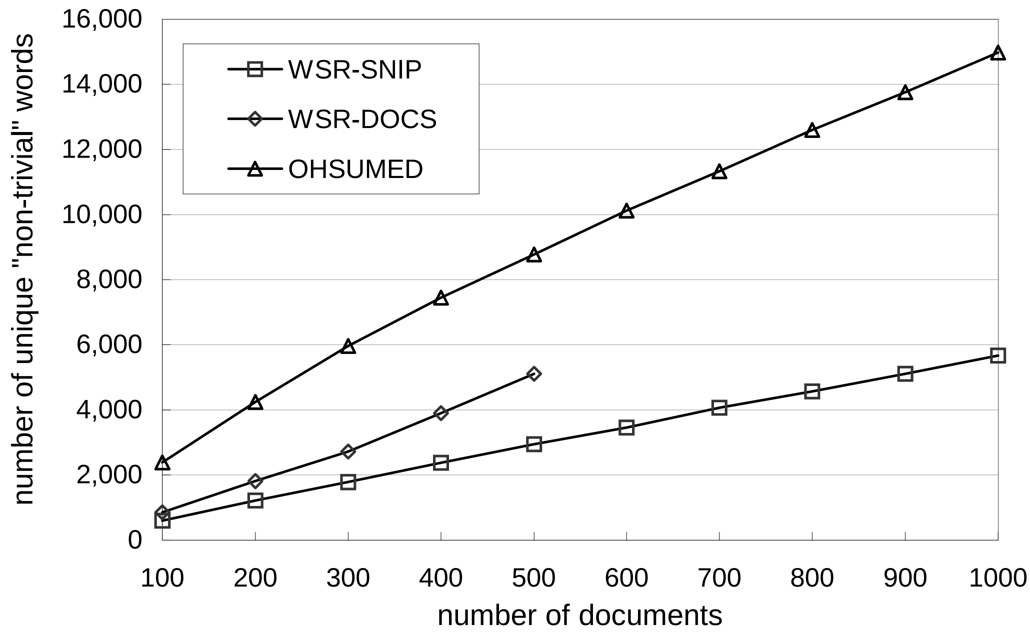


Figure 4.22: Number of unique words in the three document collections

The number of unique "non-trivial" words as a function of the document sets size for the three document collections. Non-trivial words are words that are not stopped and that appear in a minimal number of documents in the set. We can see that although OHSUMED documents are, on average, shorter than WSR-DOCS documents, they contain a larger vocabulary of words.

Figure 4.23 presents the average number of phrases identified by the STC algorithm in a document set as a function of the document set's size for the three document collections. STC identifies all phrases that appear in at least a minimal number of documents in the set (and are not contained in another phrase). We can see that OHSUMED documents contain almost 10 times more phrases than WSR documents. It is interesting that there is little difference between the WSR-SNIP collection and the WSR-DOCS collection with respect to the number of phrases identified. This can partially be explained by the fact that STC does not create phrases that cross sentence boundaries. In HTML documents, these boundaries are partly identified via HTML tags (such as
 <p> <td> <th> and <li>). These tags are often stripped in WSR snippets, thus phrases that would not be identified in the original Web document might be found in its snippet.

The running time of the STC algorithm is linearly dependent on the number of phrases found (*e.g.*, when scoring each phrase cluster). This again is linear with the average document length, but as Figure 4.22 shows STC identifies almost 10 times more phrases in OHSUMED documents as compared to WSR documents. This helps to explain why STC is noticeably slower than k-means on the OHSUMED collection while it is only slightly slower than k-means on the WSR whole document collection.

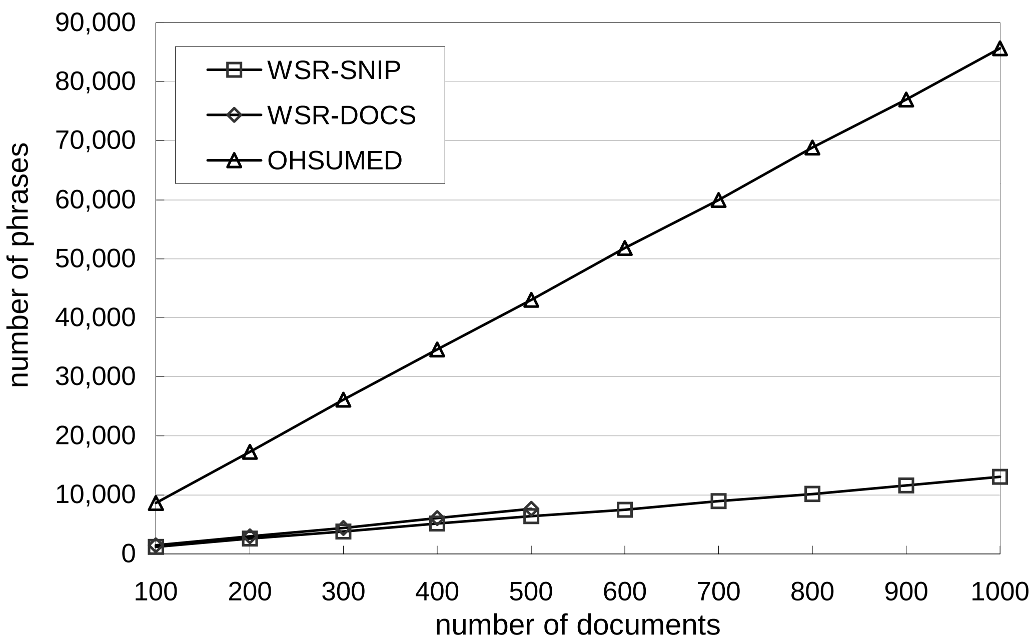


Figure 4.23: Number of phrases identified by STC in the three collections

The average number of phrases identified by the STC algorithm as a function of the document set size for the three document collections. We can see that OHSUMED documents contain almost 10 times more phrases than WSR documents. It is interesting that there is little difference between the WSR-SNIP collection and the WSR-DOCS collection with respect to the number of phrases identified. A possible explanation for this is presented in the text.

4.8 Search Engine Snippets vs. Web Documents

A major issue in the feasibility of clustering Web search engine results is whether clustering the snippets returned by the search engines can produce results that are comparable to those achieved when clustering the corresponding Web documents. Figure 4.24 (repeating the data presented in Figures 4.3 and 4.4) compares the performance of the clustering algorithms on the two WSR collections – the snippet collection and the Web document collection. In this experiment we can not use a paired statistical test; instead we use the two-sample rank test (the Mann-Whitney test). Three algorithms appear to perform similarly on the two document collections – Buckshot, GAVG and k-means. Two algorithms – Fractionation and single pass – perform better on the snippet collection (the Fractionation difference is statistically significant – $p < 0.05$; for Buckshot – $p < 0.10$), while STC perform better on the Web documents collection ($p < 0.10$).

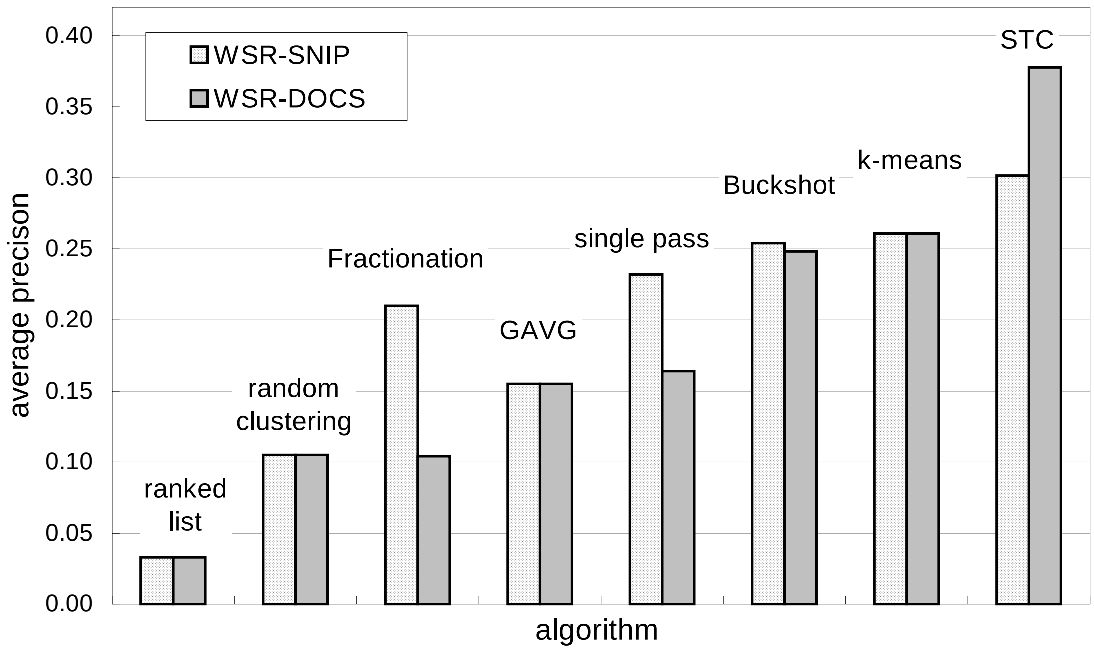


Figure 4.24: Clustering snippets vs. Web documents

The average precision for the results of the six clustering algorithms, given the 10%-viewed user model on the two WSR document collections. Most algorithms show no degradation in quality when clustering snippets (Fractionation and single pass actually perform better on snippets). STC shows 20% degradation in quality when clustering snippets. This is to be weighted against the large increase in speed (5-fold in terms of clustering time without taking into account the time to download Web documents).

We can conclude that the decrease in the quality of the clusters, if it exists at all, is relatively small. This is surprising as a Web document contained approximately 10 times more words on average than a snippet (see Table 4.3). One explanation is that the snippets represent attempts by the search engines to extract meaningful words and phrases from the original documents. Therefore the snippets contain phrases that help in the correct clustering of the document, and do not contain some of the “noise” present in the original documents that might cause misclassification of the documents. The fact the STC algorithm was the most effected by the shift to the snippets collection might indicate that search engines succeed in extracting meaningful words from documents more than they succeed in extracting meaningful phrases.

Finally, it should be noted that these results intimate earlier findings that showed that cluster quality is not adversely affected by truncating the vector representation of documents in standard IR collections [Schütze and Silverstein, 97].

4.9 Summary

In this chapter we have empirically evaluate the STC algorithm. We have evaluated the "clustering quality" of STC as compared to five other clustering algorithms on three text collections. The clustering quality was compared using two quality-evaluation approaches: the IR approach and the "merge-then-cluster" approach. We tried to use varied evaluation methods as each can easily be criticized. The consistency of the results across the different evaluation metrics and text collections is encouraging and allows us to draw cautious conclusions.

We have shown that STC, k-means and Buckshot are the highest performing clustering algorithms on our text collections. Of these, STC seems to perform the best on the WSR collections, while k-means was slightly superior on the OHSUMED collection. Buckshot appears to consistently come in third place. The other three clustering algorithms – Fractionation, GAVG and single pass performed rather poorly by our evaluation metrics, possibly because of sensitivity to parameter settings (halting criteria and thresholds) and poor performance with many outliers.

We have investigated the importance of overlapping clusters and of the use of phrases for STC by evaluating hobbled versions of the STC algorithm without these features. We have showed both to be extremely important in achieving high quality clusters. We have also demonstrated that allowing overlapping clusters can be beneficial to other clustering algorithms as well.

After evaluating the STC algorithm we have explored whether phrases can also be beneficial to other clustering algorithms. We have shown than using phrases does increase the clustering quality of vector-based algorithms. The increase in performance was most apparent for the GAVG, Fractionation and single pass algorithms, which performed poorly when using only single words, but performed comparably to the other algorithms when using phrases.

Next, we have investigated the performance of the STC algorithm using different phrase identification methods: suffix tree phrases and n-grams as well as using maximal frequent sets at multiword features. We concluded that in terms of clustering quality, the longer suffix tree phrases show little advantage over short 2-word phrases (2-grams). The main advantage of using suffix tree phrases is in their descriptive power – longer phrases are much better at describing the contents of a cluster to the user. Frequent sets were shown to produce clusters of comparable quality for the WSR-SNIP collections. However on the WSR-DOCS and the OHSUMED collections, STC outperformed FSC, demonstrating the significance of the adjacency and order information in phrases.

We have shown that the execution time of STC is comparable to that of k-means, being faster on the short documents of the WSR-SNIP collection, and slightly slower on the longer documents of the other two collections. Finally, we have examined whether when clustering search results, snippet clustering was reasonable approximation to document clustering. We concluded that the decrease in the quality of the clusters, if it exists at all, is relatively small.

Chapter 5

GROUPER I – FIRST IMPLEMENTATION ON THE WEB

5.1 Introduction

Grouper is, to our knowledge, the first post-retrieval document-clustering interface to a Web search engine. It is integrated into the HuskySearch meta-search service as one of three possible interfaces, the others being a ranked list and a presentation sorted by sites. HuskySearch (based on MetaCrawler [Selberg and Etzioni, 95]) retrieves results from several popular Web search engines, and Grouper clusters the results using the STC algorithm. Grouper is publicly available at *http://www.cs.washington.edu/research/clustering*.

In this chapter we will describe the first version of the system – Grouper I; the latest version – Grouper II – will be discussed in the following chapter. Below we describe the user interface of Grouper I, and explain key design decisions that make Grouper I fast and its clusters easy to understand. Next, we evaluate Grouper I in practice based on the logs gathered by this fully deployed system. By contrasting Grouper's logs with the logs of the standard ranked-list interface to HuskySearch, we are able to demonstrate a substantial difference in the number of documents followed, and in the amount of time and effort expended by users accessing search results through these two interfaces.

5.2 User Interface

A Grouper session starts with the user entering a query in a standard query box (Figure 5.1). The user can choose how query terms are treated (e.g., as a phrase), and can specify the number of documents to be retrieved (10 – 200) from each of the participating search engines. The system queries approximately 10 search engines and duplicate elimination reduces the size of the result set by about 50%; thus typically 70 – 1000 documents are retrieved. After all search engines have returned (or 10 seconds have passed), the main results page is displayed (Figure 5.2).

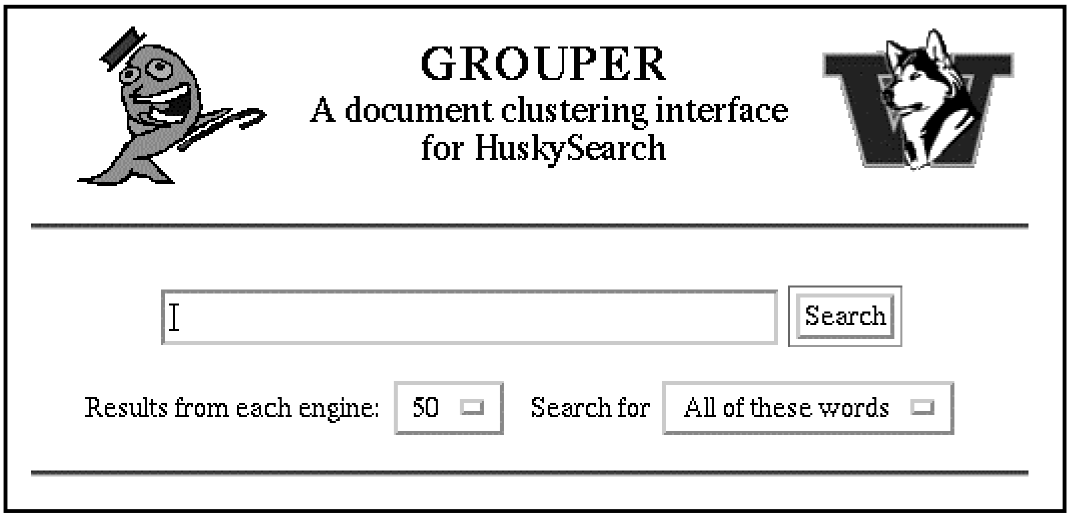
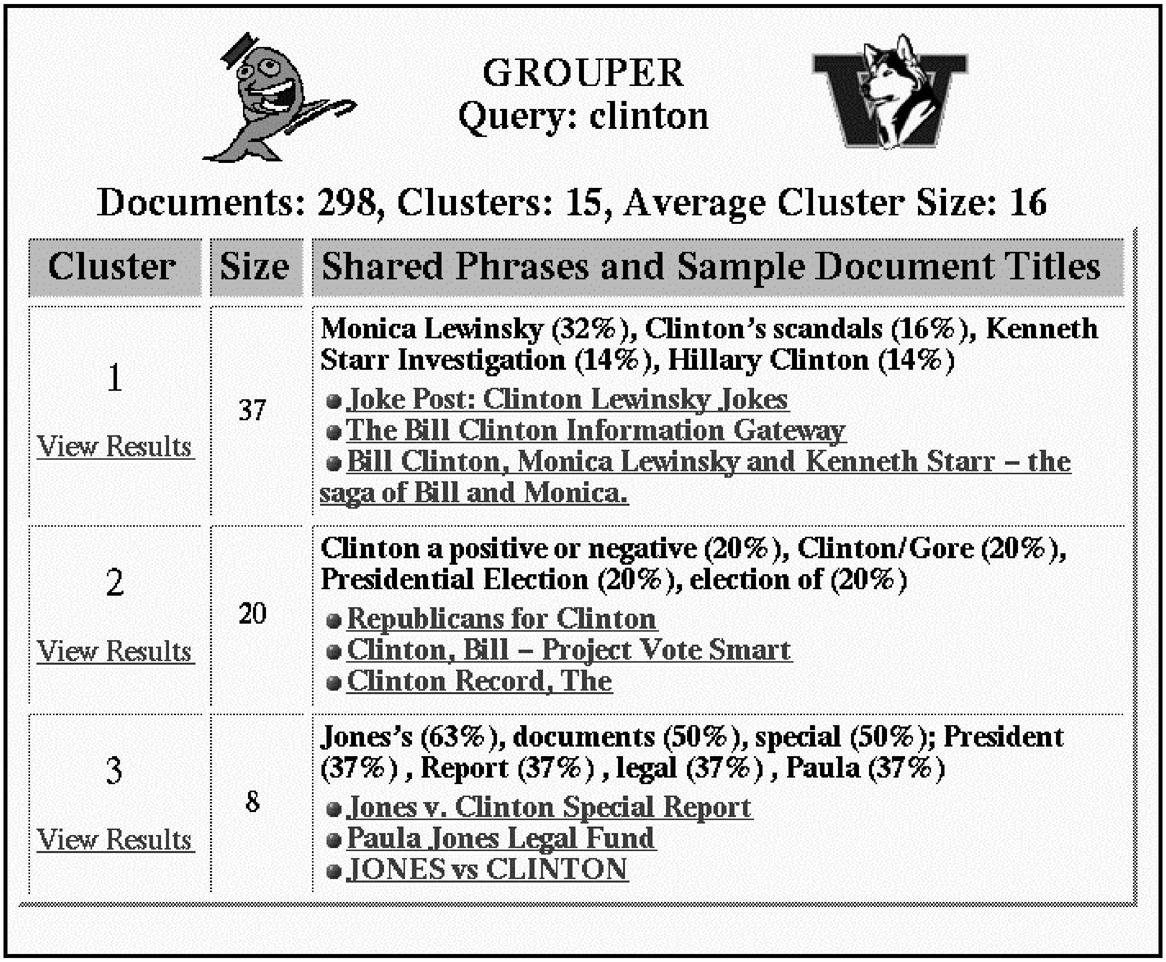


Figure 5.1: The query interface of Grouper I

Note that users do not need to enter any parameters for the clustering algorithm.

The main results page displays the number of documents retrieved and the number of clusters found. The clusters are presented in a large table – each cluster in a single row referred to as the *summary* of the cluster. The clusters are ordered by their scores, which are estimations of their coherence. A summary of a cluster includes its size (the number of documents it contains), and attempts to convey the content of the documents in the clusters by displaying *shared phrases* and *sample titles*. Shared phrases are phrases that appear in many documents of the cluster. The numbers appearing in parenthesis after each phrase indicate the percentage of the documents in the cluster that contain the phrase. Titles of three sample documents are also displayed in the summary of each cluster.

Figure 5.2 presents the main results page of Grouper I as a result of the query “clinton” (performed on May 4th 1999), which retrieved 298 documents. The system created 15 clusters, but in the interest of clarity only the first three are shown. The first cluster (containing 37 documents) contains documents pertaining to the Clinton-Lewinsky scandal and is characterized by four phrases: “Monica Lewinsky”, “Clinton’s scandals”, “Kenneth Starr investigation” and “Hillary Clinton”. The percentage in parentheses, following each phrase, indicated the portion of the documents in the cluster that actually contain that phrase. The second and third clusters relate to the presidential candidacy of vice president Al Gore and to the Paula Jones scandal, respectively.



GROUPER
Query: clinton

Documents: 298, Clusters: 15, Average Cluster Size: 16

Cluster	Size	Shared Phrases and Sample Document Titles
1 View Results	37	**Monica Lewinsky (32%), Clinton's scandals (16%), Kenneth Starr Investigation (14%), Hillary Clinton (14%)** • Joke Post: Clinton Lewinsky Jokes • The Bill Clinton Information Gateway • Bill Clinton, Monica Lewinsky and Kenneth Starr – the saga of Bill and Monica.
2 View Results	20	**Clinton a positive or negative (20%), Clinton/Gore (20%), Presidential Election (20%), election of (20%)** • Republicans for Clinton • Clinton, Bill – Project Vote Smart • Clinton Record, The
3 View Results	8	**Jones's (63%), documents (50%), special (50%); President (37%) , Report (37%) , legal (37%) , Paula (37%)** • Jones v. Clinton Special Report • Paula Jones Legal Fund • JONES vs CLINTON

Figure 5.2: The main results page

The main results page for the query "clinton". Each row in the table is the *summary* of a cluster – an attempt to convey the content of the documents in the cluster. It includes the size of the cluster, *shared phrases* – phrases that appear in many documents of the cluster, and up to three *sample titles* of documents in the cluster. The numbers appearing in parenthesis after each phrase indicate the percentage of the documents in the cluster that contain the phrase. In the example above only the first three clusters are shown.

In addition to describing a cluster using its phrases, a cluster's summary also includes words that appear frequently within the documents of the cluster. A semicolon separates these words from the phrases of the cluster. In the third cluster in our example, the words "president", "report", "legal" and "Paula" appear after a semicolon as they were found frequent among the documents of the cluster. We present up to four non-stopped words that appear in at least 30% of the documents of the cluster, and that do not appear in any of the shared phrases of the cluster nor are they query terms. Note that there is a major difference between the shared phrases and these frequent words. Shared phrases define the phrase clusters that were merged to create the cluster; therefore a phrase indicates that all the documents containing it are in the cluster. Frequent words, on the other hand, are detected after a cluster has been created. Therefore, documents containing a frequent word might be missing from that cluster.

The last row in the clusters' table is marked “Miscellaneous” (not shown in Figure 5.2), and includes all the documents that do not appear in any cluster. It is often a large set of documents (at times up to 33% of all the retrieved documents), and usually contains many documents whose snippets are totally meaningless (as frequently occurs on the Web). It may also contain relevant and interesting documents that have little in common with other retrieved documents.

If a summary of a cluster indicates to the user that the cluster is of interest, the user can take several actions. If a certain title seems promising, the user can go directly to that document by clicking on the title. Alternatively, the user can click on the "View Results" links to go to the cluster's page (Figure 5.3), which presents the snippets of all the documents in the cluster in a ranked list.

GROUPER
Query: clinton
Cluster 1

[Back to Summary]
[Refine Query Based On This Cluster]
Number Of Documents In Cluster: **37**
Phrases: **Monica Lewinsky (32%), Clinton's scandals (16%), Kenneth Starr Investigation (14%), Hillary Clinton (14%)**

Documents in the cluster:

Joke Post: **Clinton** Lewinsky Jokes
(Government > U.S. Government > Executive Branch > **Clinton** Administration > Independent Counsel Investigations > Kenneth Starr Investigation of President and Hillary Rodham **Clinton** > Subjects of Inquiry > Monica Lewinsky > Humor)
216 (1 / 39), *http://jokepost.com/clinton_lewinsky.html* (Yahoo: 1)

The Bill **Clinton** Information Gateway
Stats Updated 28 February 1999 Information about President William Jefferson (Blythe) **Clinton**, Mrs. **Clinton** (Hillary Rodham), Vice President Albert Gore, Jr., Kenneth Starr, Monica Lewinsky and other key figures in the troubled **Clinton** presidency http://www.ncal.verio.com/~suntzu75/clntgate.htm
483 (0 / 1),
http://search.excite.com/relocate/sr=webresult|ss=bill+AND+clinton+AND+information+AND+gateway|
(Excite: 2)

President **Clinton**, Monica Lewinsky and Kenneth Starr – the saga of Bill and Monica
Bill **Clinton**'s second term is already being overshadowed by scandal, we wrote in January 1997. The shadow of scandal has lengthened ever since.
557 (0 / 1), *http://ww3.economist.com/president-clinton* (AltaVista: 24)

New York Daily News Online | News and Views | Scandal Sheet | Starr Defends Sc
Independent counsel Kenneth Starr told a judge that some news leaks about his investigation of President **Clinton** and Monica Lewinsky attributed
528 (0 / 1), *http://www.nydailynews.com/1999-01-07/News_and_Views/Scandal_Sheet/a-15768.asp*
(AltaVista: 10)

Figure 5.3: The page of a cluster

The page of the first cluster created for the query "clinton". This cluster contains 37 documents pertaining to the Clinton-Lewinsky scandal and is characterized by four phrases: "Monica Lewinsky", "Clinton's scandals", "Kenneth Starr investigation" and "Hillary Clinton". Here we see the snippets of the first four documents of the cluster, as returned by the search engine.

Finally, after viewing a certain cluster, the user may want to retrieve additional similar documents. Several search engines (*e.g.*, Excite) allow the user to use a certain document as a seed for a new search (the "more like this" option). However, this method does not allow the user to control the search being made. We wanted to give the user the ability to find documents that are related to the cluster of interest, but allow the user to control the exact search that is being done. The "Refine Query Based On This Cluster" option in Grouper is designed to do precisely that (Figure 5.4) – the user can use the phrases that characterize the cluster to reformulate the query. To our knowledge, Grouper is the only search engine that suggests multi-word phrases to the user for query refinement. Looking at the logs of the system, we see that in 80% of the cases where users used the “Refine Query” option, they used the phrases suggested by the system, and 41% of these were multiword phrases. These numbers suggest that users find this feature beneficial in practice.

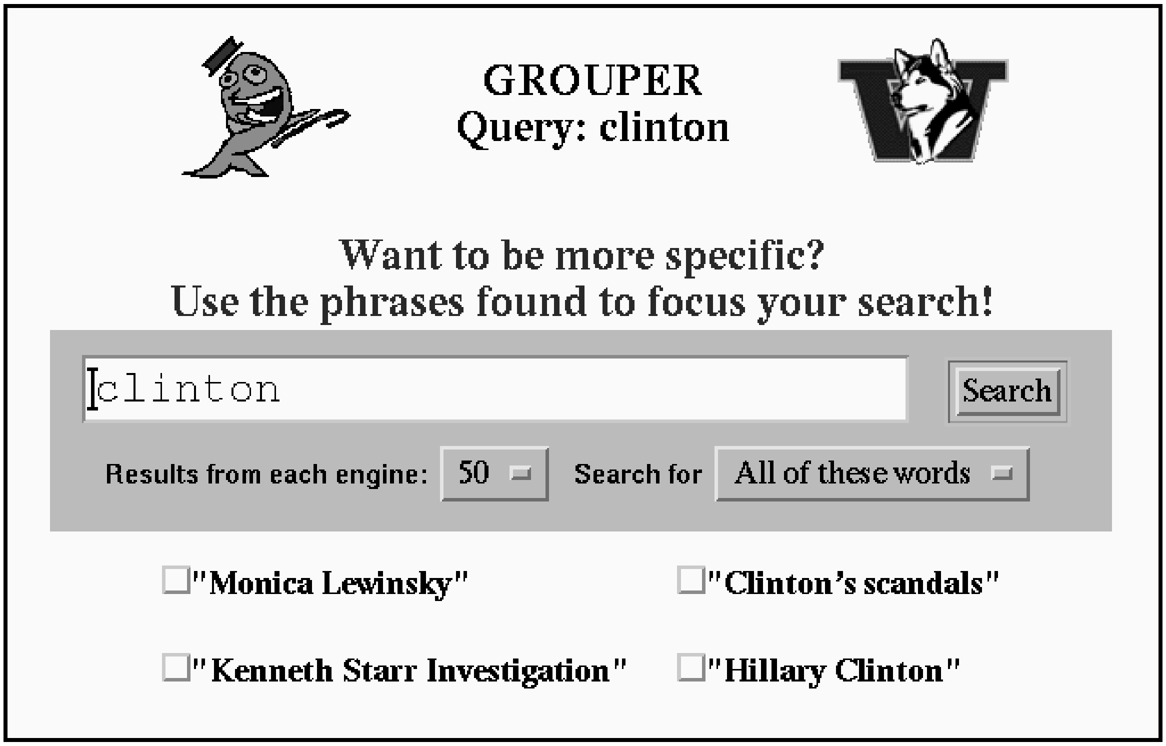


Figure 5.4: The query refinement page

The query refinement page for the first cluster of the query “clinton”. The user can reformulate the query using the phrases that were identified in the cluster. Clicking on a checkbox associated with a phrase will add it to the query box.

The Grouper search service has been running in its current format since July 1998, and receives approximately 65 queries a day. We have been logging all activities on this search engine and have logged to date more than 15,000 queries.

A noteworthy characteristic of STC is that the users do not need to input the required number of clusters. The clusters are ordered by their coherence scores, and they gracefully degrade from coherent clusters at the top of the cluster list, to clusters that are characterized by a single phrase or word at the bottom of the list (these are clusters that include a single phrase cluster). These *index clusters* (single-phrase clusters) do not necessary represent semantically coherent topics, but rather function as an index to the retrieved document set. They are useful to the user when no earlier cluster appears to be of interest. This situation occurs when there are no “natural” clusters in the retrieved document set or when the user’s interest is narrow and is overshadowed by more dominant themes among the retrieved documents. The system’s logs showed that in 21% of the sessions users followed documents from index clusters, showing the usefulness of this type of clusters.

5.3 Design Decisions

In this section, we shall review some of the design decisions that guided us in the implementation of Grouper. We divide these design decisions into two categories: those that address the speed issue and those that make the clusters easily browsable (*i.e.*, as comprehensible as possible to the users).

5.3.1 Speed

Once a user query has been issued, Grouper can cluster either the snippets returned by the search engines or the Web documents that correspond to these snippets, after downloading them from the Web. The former is faster, but the clustering quality of the latter is higher since more information is present. In the previous chapter we have investigated this tradeoff and found the degradation in the quality of the clusters produced by clustering snippets to be moderate (20% for STC using the average-precision metric). Grouper therefore, by default, clusters the returned snippets, allowing fast interaction with the user. For users willing to wait, Grouper also has a “Quality Search” option that will download the original documents off the Web and use them in the STC algorithm. This option is much slower (1-5 minutes as opposed to few seconds), but will generally produce better results.

Because STC is incremental, it can use the "free" CPU time while it is waiting for snippets to arrive over the Internet. Therefore, the algorithm could produces clustering results immediately after the arrival of the last document, whereas a non-incremental algorithm would only start its computations at that point. Because the non-incremental version of STC is more efficient than the incremental one, this speed difference would be noticeable only when using a slow network connection (*e.g.*, over a modem). As Grouper is connected to the Internet using a T1 connection we chose to implement the more efficient non-incremental version.

The STC algorithm used in Grouper is fast, as it is linear in the number of documents. In addition, we used several implementation techniques to make it even more efficient in practice. STC performs a large number of string comparisons; to do this efficiently, each word is transformed into a unique integer and thus faster integer comparisons could be used. To allow efficient calculation of document overlap between phrase clusters, the documents of each phrase cluster were encoded as a bit-vector.

A word-based suffix tree typically exhibits a large variability in the branching factor of its nodes. Table 5.1 shows the average branching factor of the suffix trees for the WSR-DOCS collection. The table displays the percentage of nodes having a branching factor in a given range (suffix trees for the WSR-DOCS collection have on average 29,000 internal nodes). Note that the single node (not 1% of the nodes) with a branching factor of more than 1000 is the root of the tree. Because of this variability in branching factors, different tree implementations were used for nodes of different branching factors. Nodes with a large branching factor (>50 – such as the root) were implemented using hash tables. Nodes with a small branching factor (<10 – the vast majority of the nodes) were implemented using sorted lists. Nodes with intermediate branching factors (10-50) were implemented using sorted trees.

Table 5.1: The variability of the branching factor of a suffix tree

branching factor:	2-10	11-20	21-30	31-40	41-50	51-100	101-200	>1000
number of nodes	55.5%	26.9%	8.4%	4.2%	2.3%	1.6%	1.1%	1

The average branching factor of suffix trees nodes for the documents in the WSR-DOCS collection. The table presents the percentage of nodes (out of approximately 29000) having a branching factor in a given range. Note that the single node with a branching factor of more than 1000 is the root of the tree (the entry in the last column of the table indicates a single node, not 1% of the nodes). Because of this variability in branching factors, different tree implementations were used for nodes of different branching factors.

Additional speedup was achieved by realizing that we were not interested in all possible phrases. Phrases that begin or end with a word appearing in our stoplist have the same semantic meaning without these stopped words, and therefore these stopped words should be stripped (for example the phrase “the vice president of” should be stripped to “vice president”, while the phrase “the vice president of the US” should be stripped to “vice president of the US”). Therefore, when inserting a string into the suffix tree, all leading and ending stopped words can be discarded. Moreover, since we are only interested in phrases that appear in more than a certain number of documents, we can also strip words that do not appear more frequently than this threshold. These improvements reduced the workload by more than 50%.

The implementation techniques described above help Grouper reach the speed required for an interactive online search engine. In Figure 5.5 we present the actual time it takes Grouper to cluster the retrieved documents as a function of the number of documents retrieved. We recorded the clustering time of 4076 Grouper queries, and grouped them by the number of documents retrieved (by steps of 50). As our machine (a DEC AlphaStation 500, 333Mhz with 512M RAM) is shared among several research projects, the reported times are greatly affected by its load as well as the load on our file server, and would be improved substantially by allocating dedicated hardware to

Grouper.

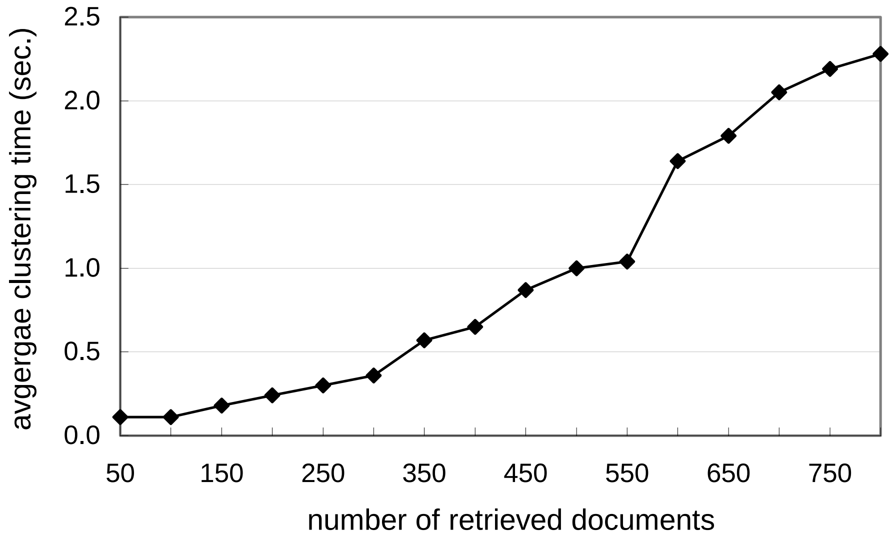


Figure 5.5: Average clustering time in Grouper

The average execution time (in seconds) of Grouper's clustering module as a function of the number documents retrieved. The reported times represent the actual delay perceived by the user (not CPU seconds). These times are greatly affected by the loads on our Web server and on our file server, and would be improved substantially by allocating dedicated hardware to Grouper.

5.3.2 Easily Browsable Clusters

As mentioned previously, it is not enough for a clustering system to create coherent clusters, but the system must also convey the contents of the clusters to the users concisely and accurately. The system is most useful when the user can decide at a glance whether the contents of a cluster are of interest.

One approach of describing a cluster is to present words that appear frequently in the documents of the cluster (the cluster's centroid) as well as the titles of selected documents [Hearst and Pedersen, 96]. As STC creates clusters based on phrases that are shared by many of their documents, Grouper can also use these phrases to describe the clusters. We have found these phrases to be extremely useful in characterizing clusters' contents.

Given a set of phrases that define an STC cluster we present them in descending order of coverage (the percentage of documents in the cluster that contain the phrase) and length (number of words in the phrase, not counting stopped words nor words appearing in the query). Since each cluster can contain many phrases, and our goal is to create a compact cluster summary, we display at most the first six phrases (the fifth and the sixth phrases are displayed only if their coverage is greater than 50%.). We have found that in many cases some of the displayed phrases are quite similar and therefore redundant. This reduces the brevity and clarity of the summary, as well as prevents additional informative phrases from being displayed. Such redundant phrases are therefore not displayed. We use three heuristics to identify redundant phrases:

1. ***Word Overlap:*** If a phrase has more than 60% of its non-stopped words appearing in another phrase *of higher coverage*, it will not be displayed. In Table 5.2, phrase 7 is not displayed because 75% of its words appear also in phrase 6, which also had a higher coverage.
2. ***Sub- and Super- Strings:*** Often, we find that STC identifies phrases that are sub-strings of other phrases (we call these *sub-phrases* and *super-phrases*). This happens when the two phrases are independent phrases in their own right (such as "united states" and "president of the united states"), or when a phrase is truncated in the snippet received from the search engine (such as "president of the united states" and "president of the united…"). A sub-phrase always has a higher coverage than its super-phrase, but the super-phrase is more specific and therefore more informative. To balance this tradeoff, we determine, for each phrase, whether it has a sub-phrase or a super-phrase among the other phrases of the cluster. If it doesn't have a sub-phrase, it is designated as a *most-general* phrase; if no super-phrase exists, it is designated as a *most-specific* phrase. We will consider phrases redundant if they are not in either of these categories. In Table 5.2, phrases 3 and 4 will not be

displayed, as they are neither most-general nor most-specific.

3. ***Most-General Phrase with Low Coverage:*** The purpose of displaying short, general phrases is to achieve high coverage. In cases were a most-general phrase adds little to the coverage of its corresponding most-specific phrase, we will not display it. Thus, we define a minimal coverage difference (we found 20% to be satisfactory) that must be attained in order for the most-general phrase to be displayed. In Table 5.2, phrase 8 will not be displayed, even though it is a most-general phrase, as it adds only 5% to the coverage of phrase 6.

Table 5.2: Determining redundant phrases

Num.	Phrase	Coverage	Most-Spec.	Most-Gen.	Selected
1	earth summit	60%	+	+	✓
2	vice president of the united states of america	30%	+		✓
3	president of the united states of	40%			
4	united states of america	50%			
5	united states	65%		+	✓
6	greenhouse gas emissions forecast	40%	+		✓
7	reducing emissions of greenhouse gas	30%	+		
8	greenhouse gas	45%		+	

The table shows the set of phrases identified in a cluster. The phrases are presented along with their coverage, whether they are most-specific or most-general and whether they are selected to be displayed. Phrases 3 and 4 are redundant, and thus will not be displayed, as they are neither most-general nor most-specific (the *Sub- and Super- Strings* heuristic). Phrase 7 is redundant as 75% of its words also appear in the phrase 6, which also has a higher coverage (the *Word Overlap* heuristic). Phrase 8 is redundant, even though it is a most-general phrase, as it adds only 5% to the coverage of phrase 6 (the Most-*General Phrase with Low Coverage* heuristic).

In the first step of the STC algorithm, documents are “cleaned” before being inserted into the suffix tree. This improves the identification of common phrases, however the readability of the text is reduced. To deal with this problem we keep the original (unprocessed) text of the documents; instead of displaying a processed and less comprehensible phrase, we display the original text that corresponds to it. For example the phrase "post retrieve document cluster" might actually correspond to the string "post-retrieval document clustering", which might be more meaningful to the user.

Another Grouper design decision was how to order the documents in a cluster. This affects both the order of the documents on the cluster's page, and the choice of the three documents that have their titles displayed in the cluster’s summary on the main results page. Two options were considered: sorting the documents based on their relevance to the query and sorting them based on the number of the cluster's phrases each contains (this is similar to the option of sorting the documents based on their similarity to the query or based on their similarity to the cluster’s centroid, as described in [Hearst and Pedersen, 96]). We chose the second approach – sorting by the number of the cluster’s phrases each document contains – because we believe this creates more informative cluster summaries.

5.4 Empirical Evaluation

The evaluation of a clustering interface is notoriously difficult, particularly in the context of Web search engines, which are used by a heterogeneous user population for a wide variety of tasks: from finding a specific Web document that the user has seen before and can easily describe, to obtaining an overview of an unfamiliar topic, to exhaustively examining a large set of documents on a topic, and more. A clustering system will prove useful only in a subset of these cases.

In the previous chapter we have evaluated the STC algorithm and compared it to other clustering algorithms. This was done using standard offline evaluation metrics, such as average-precision, which required us to develop a model of the user's use of the clustering results and to create our own relevance judgments to search results.

In this section, we choose to utilize the logs of the deployed system “in action” as the basis of our evaluation. The logs record the behavior of a large number of Grouper and HuskySearch users on queries of their choice. The Grouper logs were recorded between 15/9/98-15/11/98, and represent 3183 queries. The HuskySearch logs were recorded between 26/10/98-09/11/98, and represent 19330 queries.

5.4.1 Coherent Clusters

The first hypothesis we investigated through log analysis is that users will tend to follow documents from relatively few clusters (Figure 5.6). First, we calculated the average number of followed clusters (clusters from which a user had followed documents) as a function of the number of documents followed thus far in the session. Next, we compared this to the number of followed clusters if a random clustering had been created (using the same number of clusters and cluster size distribution).

It can be argued that by presenting the documents organized in clusters, we bias the user to follow multiple documents from the same cluster. We therefore performed the following experiment as well. We recorded which documents HuskySearch users followed. Next, we clustered the documents presented to these users and, assuming the same documents would have been followed, we calculated the average number of clusters that would have been followed in these HuskySearch sessions. To provide a benchmark for STC's performance on the HuskySearch data, we compared STC with the k-means clustering algorithm. We have compared STC with additional clustering algorithms and found similar results, with Buckshot performing slightly better than k-means and Fractionation, GAVG and single pass slightly worse.

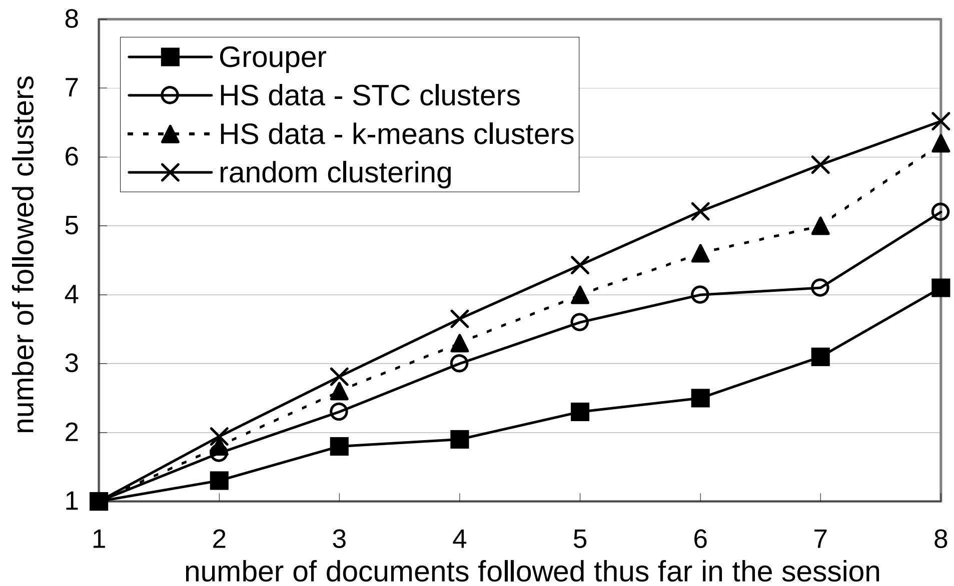


Figure 5.6: The average number of clusters followed

The average number of followed clusters as a function of the number of documents followed thus far in the session. We compare the number of clusters followed in Grouper sessions to the number of followed clusters if a random clustering had been created, and show that Grouper is substantially better than random. In addition, we present the calculated number of clusters followed using HuskySearch data: the documents retrieved in HuskySearch search sessions are clustered using both STC and k-means; assuming that the same documents would have been followed, we calculate the number of followed clusters. We see that STC produces slightly more coherent clusters than k-means using this analysis.

We had hypothesized that the user would visit few clusters in Grouper, but the numbers were not as low as we had hoped (*e.g.*, users viewing seven documents visit three clusters on average). This result can be interpreted in several ways. It might indicate that even when users follow many documents, they typically seek an overview of the retrieved results and not necessarily an exhaustive review of all the documents on a specific subtopic. Alternatively, the results might indicate that our clustering algorithm (using the documents' snippets) falls short of the ideal goal of creating a single cluster corresponding to the user's information needs.

It should be noted that the procedure detailed above is in fact a novel method for comparing clustering algorithms; the novelty being in the construction of the document collection. Here, we define document sets based on the retrieved results of a search engine. Relevance judgments are defined by assuming relevant all documents the users choose to view using a ranked list presentation. Needless to say, these relevance judgements are not fully accurate (the users might view irrelevant documents), nor complete (the users might not have viewed all relevant documents), but so are relevance scores assigned by humans. The big advantage of this method is that algorithms can be tested on a very large scale (huge number of document sets defined by a large number of users). Moreover, the document sets are very varied and are constructed from real data (the data that is of interest to the users of the system). Data collections with relevance judgements assigned by humans, on the other hand, are typically small and require vast resources to construct.

5.4.2 Comparison to a Ranked List Display

In this section, our goal is to compare the Grouper clustering interface to the traditional ranked-list interface available for HuskySearch. HuskySearch and Grouper are identical (including the machines they run on) with the exception of their result presentation.

Having recorded user behavior on both Grouper and HuskySearch, we would ideally report whether users were able to find more relevant information, and do so more efficiently, using Grouper's clustering interface as compared with the ranked-list presentation in HuskySearch. However, since in contrast with TREC experiments we do not know the exact information need or the set of relevant documents, we cannot rely on the standard notions of precision and recall. Instead, we decided to use the following metrics in our evaluation, each of which is subject to multiple caveats which we mention below:

1. **Number of documents followed:** we recorded the number of documents “clicked

on" by users as an indication of the "amount" of information users were able to retrieve (Table 5.3). However, clicks are a coarse measure of the amount of information obtained; one click may indicate mild interest whereas another may indicate that the user "struck gold". Worse, people may be seduced by a snippet into clicking on a document that is of no interest. Finally, users may obtain valuable information directly from snippets or cluster summaries without any clicks.

2. **Time spent:** as a rough measure of search efficiency, we recorded the amount of time users spent traversing the results set (Figure 5.7). The main problem with this measure is that the time recorded is the *sum* of time spent in network delays, in reading documents, and in traversing the results presentation (ranked list or clusters). Our server-based experimental apparatus cannot decompose the time measured into its components.
3. **Click distance:** as another rough measure of search efficiency, we recorded the distance between successive user clicks on the document set (Figure 5.8). While in the ranked-list presentation, this notion has an unambiguous interpretation (the distance between the first document in the list and the tenth document is nine), it's not immediately obvious what is the corresponding notion in the cluster presentation. Specifically, when a user skips a cluster, what is the distance traversed? We discuss our distance function below.

We acknowledge that in the experiments described below, we are comparing the behavior of two distinct user populations – HuskySearch users and Grouper users. Nevertheless, the data we collected shows substantial differences in how people use the two interfaces. We plan to follow up with a carefully controlled user study that will complement the log analysis and help to test hypotheses as to *why* the observed differences arise.

Looking at the number of retrieved documents that were followed by the user, we note a considerable difference between the two systems. In HuskySearch users followed only 1.0 documents on average, whereas in Grouper, users followed 40% more. Table 5.3 presents the percentage of sessions in which the user has followed a certain number of retrieved documents for both systems. All sessions resulting in 8 or more followed documents are presented in a single column (the “8+” column). The data shows that the percentage of sessions in which users do not follow even a single document is smaller in Grouper (46%) than in HuskySearch (53%), while the percentage of sessions in which users followed multiple documents is higher in Grouper.

Optimistically, Table 5.3 supports three possible hypotheses: First, it might be easier to find interesting documents using Grouper (as fewer sessions result in no document being followed). Second, once an interesting document has been located, Grouper seems to do a better job in helping the user find additional interesting documents. And third, users might prefer a clustering interface such as Grouper when faced with tasks where several documents are required. As mentioned earlier, a user study is needed to test these hypotheses.

Table 5.3: Number of documents followed in Grouper and HS sessions

num of documents followed	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8+**
% of HuskySearch sessions	53.0	26.9	8.4	4.2	2.3	1.6	1.1	0.7	1.9
% of Grouper sessions	46.0	25.2	10.2	6.0	3.9	2.4	1.8	1.4	3.2

The percentage of sessions in which users have followed a certain number of retrieved documents in Grouper and in HuskySearch. All session resulting in 8 or more followed documents are presented in a single column (the “8+” column). In HuskySearch, users followed two or more documents in 20% of the sessions, while in Grouper they did so in 29% of the sessions.

The second metric we use to compare the two systems was the time the user spent on each document followed. When a user followed a document, the request was actually routed through our server and we logged the arrival times of these requests. The time spent on a followed document is measured as the time between a user’s request for a document and the user’s previous request (or, for the first followed document, the time the results page was sent to the user). The measured time therefore includes the time to download and view a selected document and the time to find the next document of interest. We exclude from the analysis all sessions in which more than 200 seconds elapsed between the following of consecutive document (approximately 10% of the sessions) as the user might have abandoned the search task for a while. The time spent on each document as a function of the number of the documents followed thus far in the session is presented in Figure 5.7.

From Figure 5.7 it can be seen that the average time per document for the first three followed documents of a session is lower in HuskySearch, but the Grouper interface is apparently faster for the rest of the followed documents of the session.

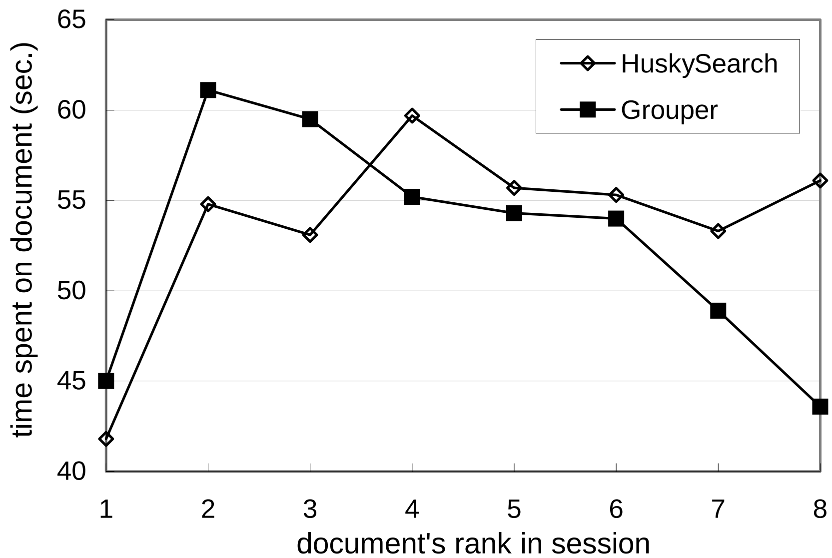


Figure 5.7: Average time per document in Grouper and HuskySearch sessions

The average amount of time spent on each document followed, as a function of the document’s rank (the number of documents followed thus far in the session).

While speed appears to be an advantage for Grouper when multiple documents are followed, some caveats are in order. In the analysis above we assume the time to download a document and the time to view it are independent of the system used and of a documents' rank in the session, and therefore these factors should average out across all measurements. Unfortunately, we cannot validate this empirically in our system. Another possible problem in interpreting this metric is that it is not clear that shorter time is better. It might indicate that an interesting document was easier to find, but on the other hand, it might indicate that the user did not find the document interesting enough to dedicate much time to it. Again, the logs do not give us enough information to discriminate between these possible interpretations.

The third metric used to compare the two systems was click distance, an attempt to estimate the cost to the user of finding the next interesting document. We assume only documents followed are beneficial to the user; snippets that are scanned but not followed represent wasted efforts. Likewise, the user is assumed to gain nothing from the structure of the clusters. The distance between clicks on a ranked list is simply the number of snippets between the two clicks (or proceeding the first click). In the cluster interface, we assume the user does not scan through the snippets in clusters that are skipped, but in any cluster visited all of the snippets are scanned. For example, if the user clicks on the second document of the first cluster (which contains 20 documents), skips the second cluster and then clicks on the fifth document of the third cluster, we assume she scanned 22 snippets between these two clicks (18 in the first cluster and 4 in the third). In the clustering interface there is the additional cost of skipping clusters. This should be added to the measure presented above as they both represent wasted effort.

This data is presented in Figure 5.8. The "Grouper-snippets" line represents the number of snippets skipped. The "Grouper-total" line represents the total click distance, which is the sum of the number of snippets and clusters skipped. We assume the "cost" of scanning a summary of a cluster is equal to that of scanning a snippet. Of course, alternative assumptions can be made, and it is easy to see from the graph how these alternative "Grouper-total" lines would look.

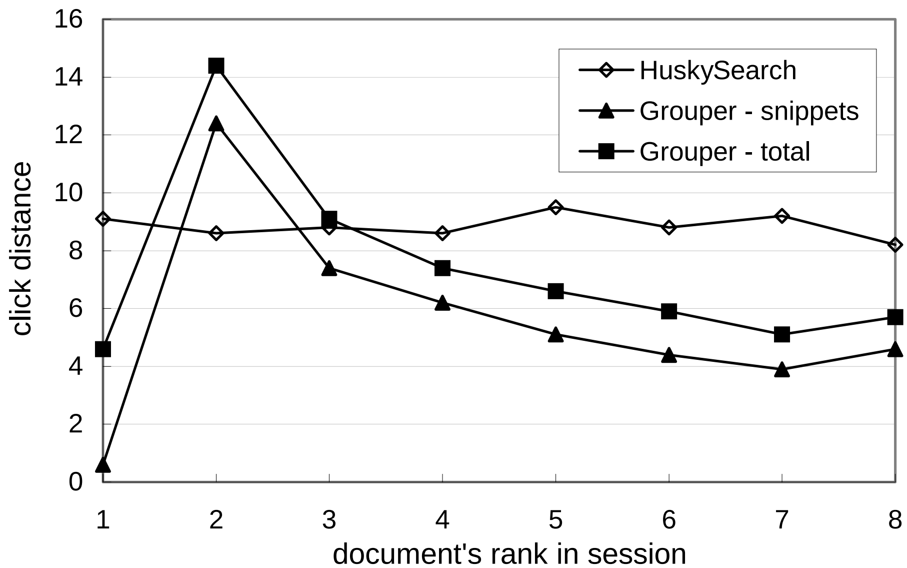


Figure 5.8: Click distance in Grouper and HuskySearch sessions

The click distance as a function of the document's rank. This metric attempts to capture the effort spent by the user to find the next interesting document. The "Grouper-snippets" line represents the number of snippets skipped. The "Grouper-total" line represents the total click distance, which is the sum of the number of snippets and clusters skipped (under the assumption that the "cost" of scanning a cluster's summary is equal to that of scanning a snippet).

The results presented in Figure 5.8 exhibit a trend similar to the results of the previous experiment – finding the first few interesting documents actually requires more "effort" in Grouper as compared to HuskySearch's ranked-list presentation, but after the first two or three documents, finding additional interesting documents appears to require less effort in Grouper. This possibly reflects the fact that the user must spend some time/effort to understand the clusters that Grouper presents, but after doing so the clusters are helpful in finding required information faster. It also supports our hypothesis that a clustering interface is not suitable for all search tasks.

5.5 Summary

In this chapter, we have presented Grouper – a fully implemented document clustering interface for Web search engine results, which has been deployed on the Web since June 1998. Grouper is integrated into the HuskySearch meta-search service, which retrieves results from several popular Web search engines and passes them on to Grouper to be clustered via the STC algorithm.

We described the user interface of Grouper I, and explain important design decisions that make Grouper I fast and its clusters easy to understand. Grouper I enabled us to carry out the first empirical assessment of user behavior given a clustering interface to Web search results. By analyzing the system's logs and comparing them to the logs of HuskySearch, we were able to show substantial difference in the usage patterns of the two systems. We showed a difference in the number of documents followed, and in the amount of time and effort expended by users accessing search results through these two interfaces.

By fielding Grouper I on the Web, we were able to identify some of its shortcomings: the results are, at times, confusing, not always useful, and not always intuitive for the novice user. Grouper II, presented in the following chapter, was designed to address these shortcomings.

Chapter 6

GROUPER II – THE NEXT GENERATION

6.1 Introduction

In this chapter we introduce the Grouper II system, the second generation of our document clustering interface to Web search results. Grouper II was designed following six months of experience with the Grouper I interface and addresses the shortcomings that were identified in Grouper I. We start by discussing some of these shortcomings. Next, we describe the Grouper II system, explain how it addresses these shortcomings, and present its user interface. In order to evaluate Grouper II we analyze its logs and contrast them with the logs of Grouper I. Finally, we describe a user study that was performed with the Grouper II system to validate its usefulness.

6.2 Shortcomings of Grouper I

Grouper I creates clusters by merging phrase clusters (a phrase and the set of documents that contain it). This is often beneficial as it reduces the fragmentation of the document set and constructs larger units relating to a coherent semantic meaning. But the merging of phrase clusters can be confusing, as it can place dissimilar documents in the same cluster.

Sometimes, Grouper I identifies phrases that are common in the collection but are of no values (*e.g.*, “This was last updated on”). These are often what we have termed *structural phrases*, as they typically relate to the structure of the documents (and its source) rather than to its content. The user may wish to disregard these phrases, but the system currently has no way of identifying them. When they are used to construct merged clusters, the result might be confusing and meaningless clusters.

Another problem with any clustering system is that “natural” clusters do not always exist in the data set. For example if one were to apply a clustering algorithm to a set of points distributed uniformly on a plane, the results would be meaningless. The optimal action in such cases is to have the system identify that natural clusters do not exist, and report that to the user. From our experience, this happens quite often in the domain of search engine results.

Our clustering interface, though simple, can be confusing to novice users. For example, we have observed that some novice users did not realize they had the option of seeing a list of snippets of all the documents of a cluster. Instead they assumed they had access only to the titles appearing in the clusters' summaries on the main page. Even those that understood the interface needed time to be able to use it efficiently. For example some users, after issuing a poor query, would explore most of the clusters, even though it was obvious from the main page that the query had to be reformulated.

We have also noticed that the number of clusters created by STC, and their sizes, increased as more documents are retrieved. Therefore, in large document sets the users have to scan many documents either because they scan many clusters or because the clusters they scan are large. As we are interested in making the system scale to substantially larger retrieved document sets, clusters should be presented hierarchically so users can navigate the results more efficiently.

In the next section we shall describe one possible solution to address the issue of poor clusters – presenting the clusters only when their quality is sufficiently high. Next we describe Grouper II, which provides a much more comprehensive solution, and discuss how it addresses the shortcomings described above.

6.3 Advertising Clusters

As mentioned previously, the clusters produced by any clustering algorithm can at times be totally useless to the user. One possible solution to this problem is to evaluate the quality of the clusters and to present them to the user only when the quality is sufficiently high. This is only a partial solution as it helps the user avoid the clustering interface when it is not helpful, but it does not attempt to increase the usefulness of the interface (as Grouper II does).

In our implementation of this approach the user's default interface is a ranked list. We cluster the retrieved documents (not displaying the results), and calculate the quality of the clustering using a very simple formula – we sum the scores (given by the STC algorithm) of the first three clusters. If the clustering quality is above a certain threshold, we display a small "Ad" at the top of the ranked list interface, suggesting the user switch to the clustering interface. An example of the Ad for the clusters generated for the query "clinton" is shown in Figure 6.1. The Ad describes the first three clusters by listing their phrases (the same phrases that are used in the cluster summaries). If the users find the Ad intriguing, they can click on it to go to the clusters' interface (or go directly to a specific cluster's page by clicking on that cluster's line in the Ad).

By analyzing the logs of HuskySearch we discovered that in 6% of the sessions users proceeded to view the clustering interface (clusters were viewed in 3539 sessions out of 58936 HuskySearch sessions during 1-10 of August, 1999). This was much higher than the 1% of the users that viewed the results sorted by site.

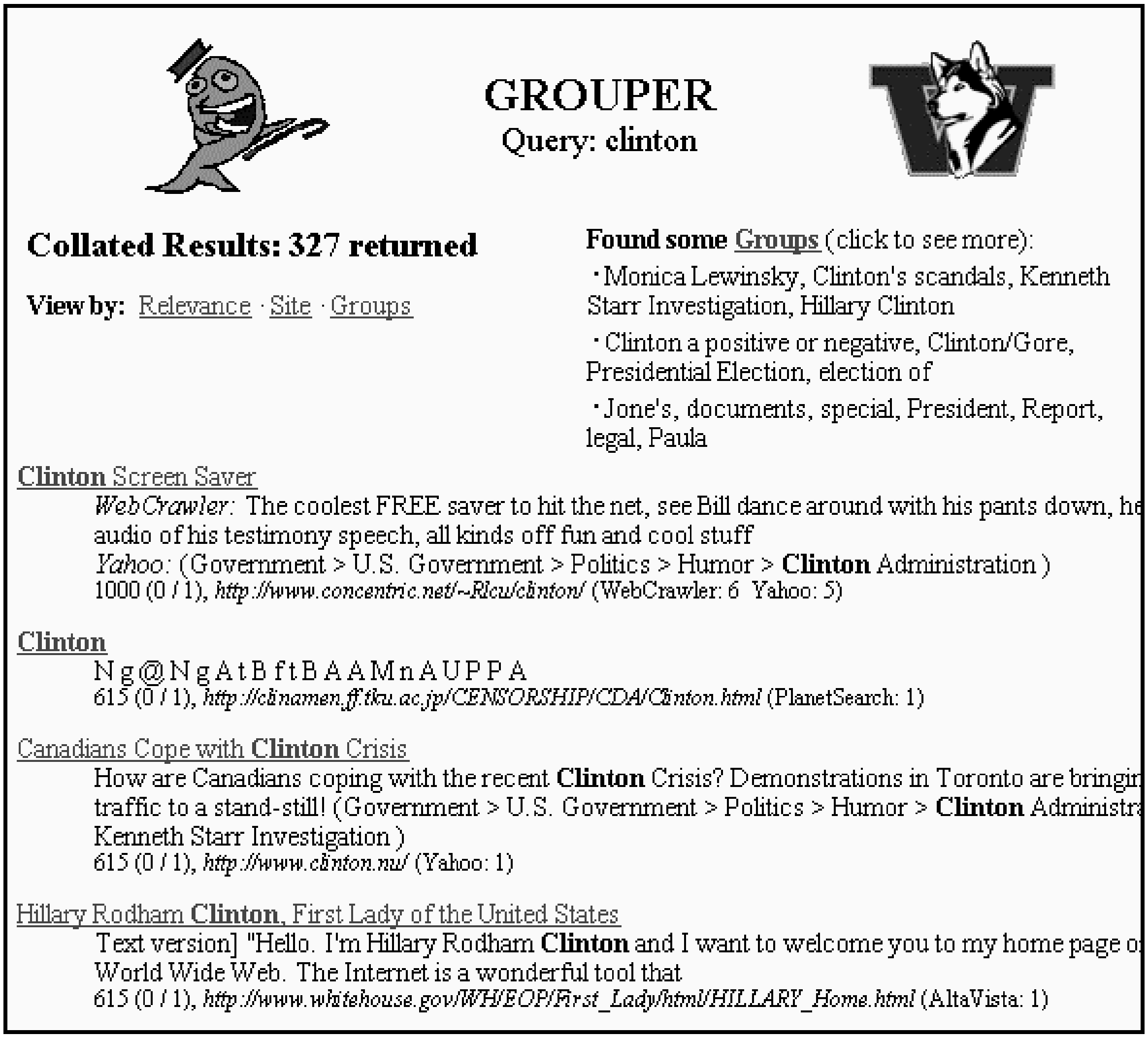


Figure 6.1: Advertising the clusters in a ranked list interface

Displaying an "Ad" for the clustering results of the query "clinton" in a ranked list interface. The Ad is shown in the top-right corner of the figure, under the title "Found some Groups". The retrieved documents are clustered (without displaying the results), and the clustering quality is calculated. If the clustering quality is above a certain threshold, we display the Ad, allowing the user to switch to the clustering interface if the clusters are intriguing.

6.4 Grouper II

Grouper II has three components that are novel compared to Grouper I:

1. **Dynamic Index:** This new interface paradigm allowing users to view non-merged phrase clusters. This view is essentially an inverted index of phrases to the retrieved document set. When using the screen space efficiently, one can present 30-50 phrases in a single screen thus saving the user the inconvenience of scrolling down the window. We believe this view is useful in a high portion of the queries (higher than the Grouper I interface) as it does not stumble on meaningless phrases – they are presented but the user can easily avoid them. In Grouper I such phrases are harder to avoid as they might be merged with other phrases (forming a merged cluster).
2. **Multiple Views:** Grouper II allows the user to view the documents in one of three interfaces: a dynamic index, clusters or a ranked list. As we have stated earlier, clusters are not useful for all information needs (neither is a dynamic index). Selecting the proper view allows the user to befit the interface to the task at hand.
3. **Interactive and Hierarchical Navigation:** Grouper II supports a hierarchical and interactive interface, similar to the Scatter/Gather interface [Cutting et. al., 92]. Users can select one or more index entries or clusters to focus on, thus define a subset of the retrieved documents. Users are then able to recursively "zoom in" on this subset (viewing it via one of the three possible views). This feature allows users to navigate much larger document sets than the previous "flat" display of Grouper I.

In the next section we describe Grouper II's user interface and detail how the phrases displayed in the dynamic index interface are selected.

6.4.1 User Interface

The query form of Grouper II is displayed in Figure 6.2. It is similar to the query interface of Grouper I (and of HuskySearch) with one addition – the user can select one of three basic display modes. The possible display modes are a dynamic index, a clustering interface and a ranked list presentation. The dynamic index interface displays phrase clusters while the "clusters" interface presents merged clusters (the "combined" option displays both the dynamic index and the merged clusters on the same screen). Note that as before the user can specify whether the system should download the documents off the Web for a more comprehensive, yet slower, analysis.

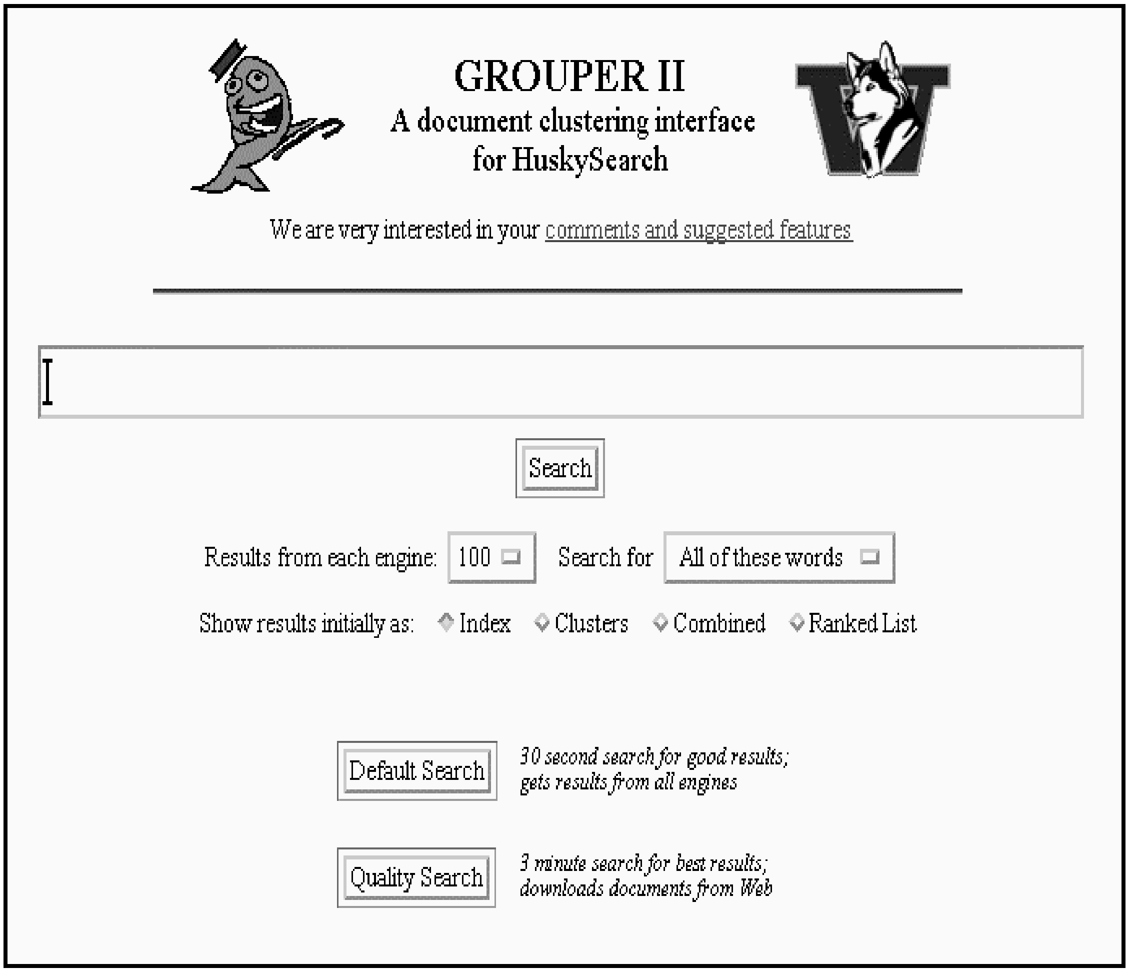


Figure 6.2: The query interface of Grouper II

The user can select one of three basic display modes in which to view the results – a dynamic index, a clustering interface or a ranked list interface. The dynamic index interface displays phrase clusters while the "clusters" interface presents merged clusters (the "combined" option displays both the dynamic index and the merged clusters on the same screen).

Figure 6.3 presents the dynamic index interface generated for the query "clinton". The dynamic index displays "interesting" phrases that were identified in the returned document set (an "all others" entry also exist, designating all documents that can not be identified by any of the phrases in the index). Each phrase displayed is in fact a phrase cluster. The number of documents in the phrase cluster appears in parentheses besides the phrase. The user can click on the checkboxes preceding the phrases to mark the phrase clusters that are of interest. Then, after selecting the next display mode, the user can “zoom in” on the subset of documents defined by the selected phrase clusters.

If the user has not yet asked the system to download the documents off the Web, she can do it at any time by clicking the “download documents” checkbox. This will download only the documents in the subset defined by the user, thus performing the more expensive analysis only on those documents deemed relevant.

At any time the user can decide that the query should be reformulate and resubmitted to the search engines. The “new query” input box is provided for that purpose. A selected phrase is automatically added to the “new query” query box, as the phrases in the dynamic index can be very helpful in defining the reformulated query.

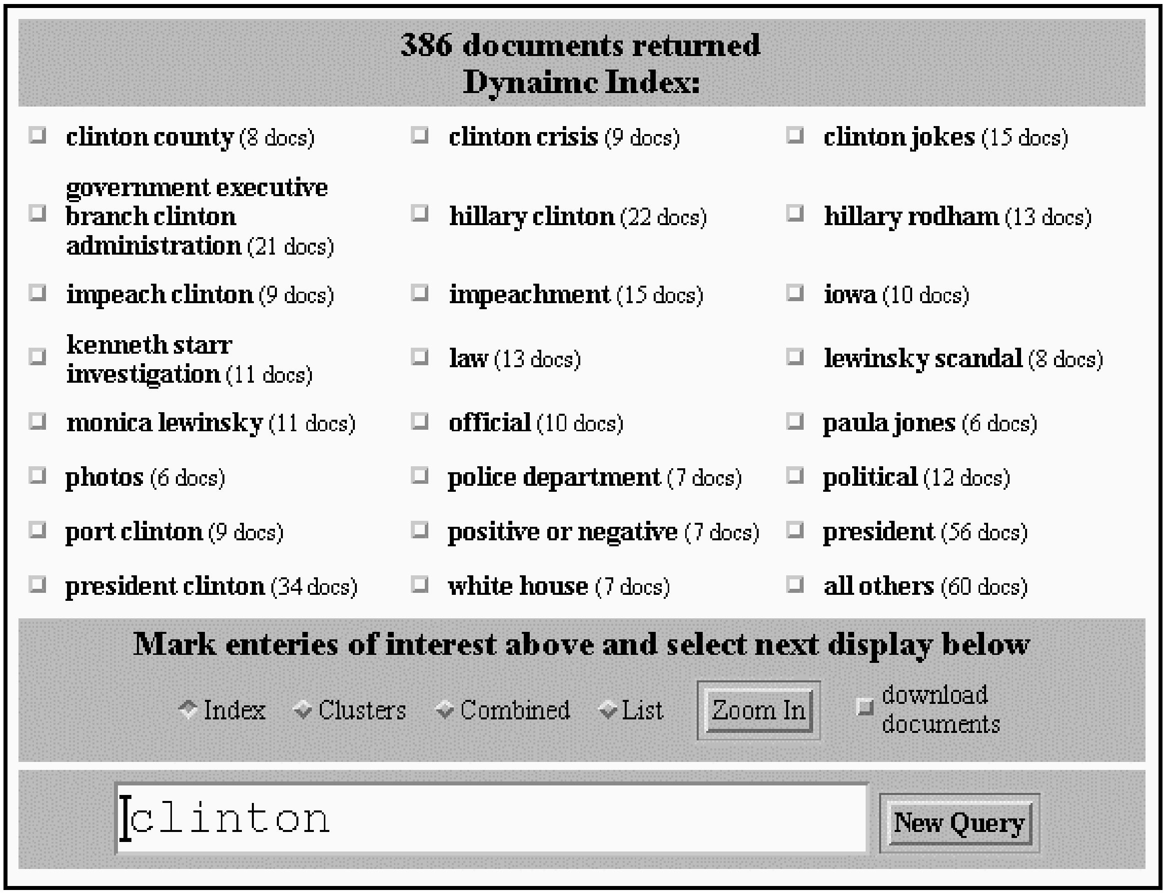


Figure 6.3: The dynamic index interface

The dynamic index interface generated for the snippets returned by the query "clinton". The dynamic index displays "interesting" phrases that were identified in the returned document set. The "all others" entry designates all documents that do not include any of the phrases in the index. The user can click on the checkboxes besides the phrases to mark the phrases that are of interest. Then, after selecting the next display mode, the user can "zoom in" on the subset of documents defined by the selected phrases.

6.4.2 Dynamic Index Generation

The dynamic index interface displays up to 35 phrases (12 columns of three, including the "all others" entry). It is generated by selecting a subset of the phrases identified by the suffix tree. We attempt to select a subset of high quality phrases (*i.e.*, phrase clusters that receive a high score in the STC algorithm), that also provide a high coverage (the percentage of documents that include at least one of the phrases) of the results set. We use a simple greedy algorithm to achieve this. First, each phrase is scored using the same phrase cluster scoring function described earlier (the score is a function of the number of documents containing the phrase, the length of the phrase, and the tfidf value of the words of the phrase – see section 3.3.2). Next, we order the phrases based on their scores. Finally, we process the phrases, in descending score order, greedily selecting a phrase if it does not overlap with any previously selected phrase by more than a certain threshold α (overlap of the phrases' document sets).

By selecting different thresholds α, different subsets of phrases will be selected. We can quantify the quality of each subset using two measures: a phrase-quality measure – the sum of the scores of the phrases in the subset – and the coverage measure – the percent of the documents that contain at least one of the phrases in the subset. There usually is a tradeoff between these two measures. Instead of fixing the threshold α in advance, we select subsets for different values of α, and then choose the best one based on these two quality measures. Currently we use the following guideline – select the subset with the highest phrase-quality measure among all the subsets with coverage of at least 80%. Needless to say, more sophisticated procedures can be adopted.

6.5 Evaluation

Grouper II was designed in light of the shortcomings we had identified in Grouper I. But was it in fact better? And how does it compare to a ranked list interface? We evaluated the Grouper II system both by analyzing its logs and comparing them to the logs of Grouper I (presented in the previous chapter), and by performing a controlled user study comparing the browsing abilities of users given a dynamic index interface and a ranked list interface.

6.5.1 Log Analysis

We analyzed the logs of the Grouper II system using the same methodology and metrics described in the previous chapter. However, several caveats are in order. Grouper II was implemented towards the end of this study. It was designed as a proof of concept – therefore its implementation was "quick and dirty" resulting in slower performance than could have been achieved with additional engineering efforts (for example each iteration of the user is a separate process and all communications between these processes is done via files on disk). Grouper II was deployed in May 1999, thus we believe that many of its users are still "experimenting" with it, rather than using it to help them search for information on the Web. Finally, due to limited support the system was often down, perhaps as much as 30% of the time.

The Grouper II logs were recorded between June 1st, 1999 and August 31st, 1999 and represent 1404 queries. We first compared the Grouper I system and the Grouper II system based on the average number of followed clusters (clusters from which a user had followed documents). This experiment was previously described in section 5.4.1 (Figure 5.6). A Grouper II cluster in this experiment is not a phrase cluster (as the user might zoom in several times on several sets of selected phrase clusters), but rather any subset of the documents that was finally viewed using the ranked list interface.

The data comparing Grouper I and Grouper II is presented in Figure 6.4. From the figure it is obvious that Grouper II does a better job in placing together the documents that seem interesting to the user.

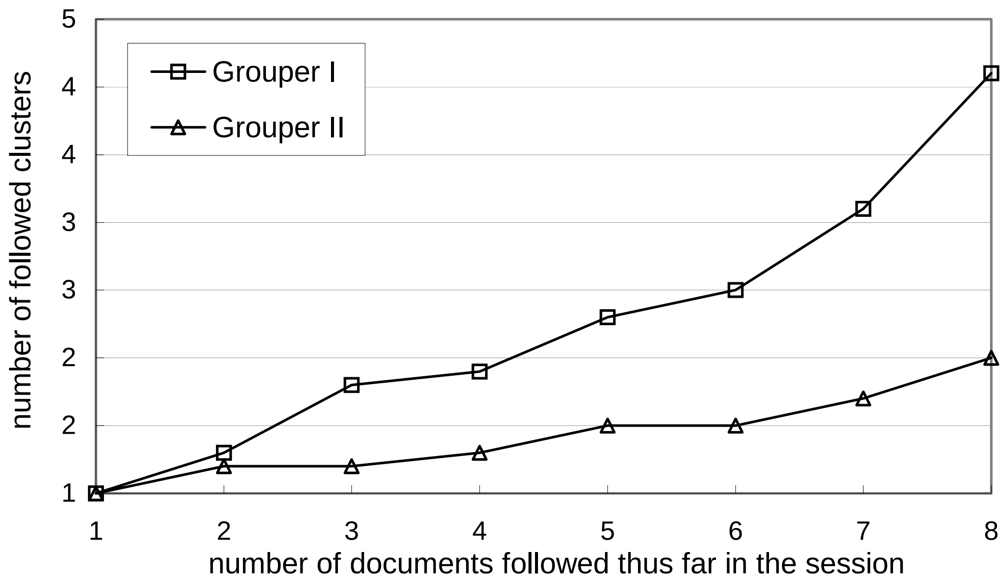


Figure 6.4: The average number of clusters followed

The average number of followed clusters in Grouper I sessions and in Grouper II sessions as a function of the number of documents followed thus far in the session. For this experiment we define a cluster in the Grouper II system as any subset of the documents that was finally viewed using the ranked list interface. It appears that Grouper II does a better job in placing together the documents that seem interesting to the user.

The second metric we used to compare the two systems was the time the user spent on each document followed. This experiment was previously described in section 5.4.2 (Figure 5.7). As before, we exclude from the analysis all sessions in which more than 200 seconds elapsed between the following of consecutive document as the user might have abandoned the search task for a while.

Grouper I and Grouper II used different default settings regarding the number of documents to retrieve from each search engine. Grouper I had the default value at 50 documents, whereas Grouper II, designed to support browsing of larger document sets, had the default value at 100. This caused Grouper I to retrieve on average 250 documents, while Grouper II retrieved approximately 450 documents. To compensate for this we subtracted from the time of the first Grouper II document the difference between the average clustering time in the two systems. We also subtracted the time Grouper II spends writing files to disk for the next iteration (as this can easily be eliminated with better implementation).

The data comparing Grouper I and Grouper II is presented in Figure 6.5. Finding the first interesting document is apparently slower in Grouper II, however finding subsequent interesting documents seems faster. Users of Grouper II might require more time to find the first interesting document because of several possible reasons: (1) it might take longer to understand the display and to decide how to proceed; (2) the average Grouper II user has less experience with the system than the average Grouper I user; (3) Grouper II often requires more iterations (several "zoom ins") than Grouper I; (4) Grouper I displays titles of documents on the main clustering page (in the clusters' summaries). Users can directly follow a document from that page. In Grouper II we do not provide that option, requiring users to first view the documents' snippets as a ranked list before following a document.

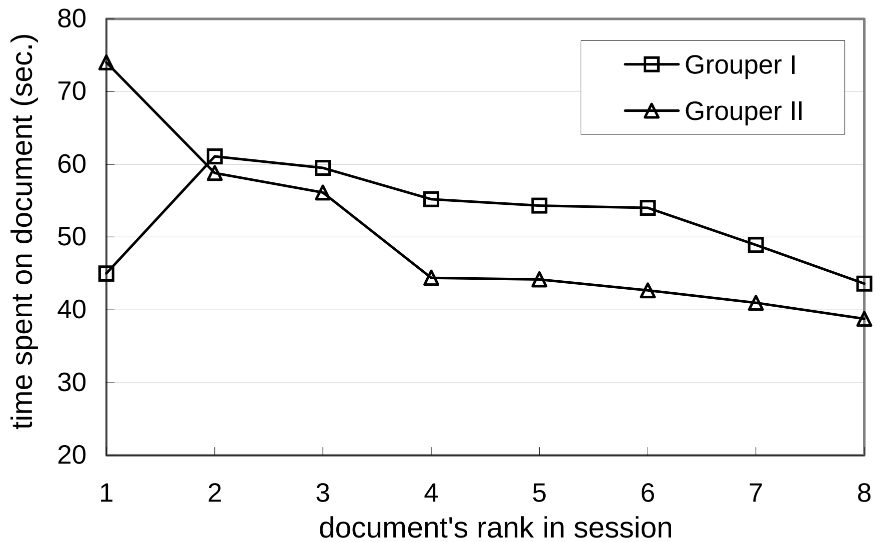


Figure 6.5: Average time per document in Grouper I and Grouper II sessions

The average amount of time spent on each document that was followed, as a function of the document's rank (the number of documents followed thus far in the session). Finding the first interesting document is apparently slower in Grouper II, however finding subsequent interesting documents is faster.

6.5.2 User Study

In a second evaluation of the Grouper II system, we performed a small-scale controlled user study in which we compared the search behavior of a group of users on specific tasks using both Grouper II and a ranked list presentation. Seven participants took part in the study during May 1999. Because of the small number of participants the results presented below are not statistically significant, but they can provide anecdotal evidence as to when an interface such as Grouper II would be more useful that a ranked list presentation. Another problem with this experiment is that all participants had substantial previous experience with a ranked list interface, however this was their first use of the Grouper II interface (they were given only 2 "warm up" queries).

The design of the user study was as follows. Twelve different assignments were defined. For each assignment we specified its title, the information need of the user and the task the user must perform. Six of the tasks required the user to find a single document relating to the information need; the other six required the user to locate as many documents as possible. We later analyzed the data separately for these two different types of information tasks. Figure 6.6 lists two sample information tasks, one of each category.

Assignment 6	
Title:	Gore's Internet statement
Information Need:	You are looking for an exact quote of the statement Gore made about his claim to the development of the Internet.
Task:	Find ONE page with the exact quote of Gore’s statements.
Assignment 10	
Title:	Melbourne’s penguins’ parade
Information Need:	You want to go to see the Penguins’ Parade out of Melbourne. You are looking for organized day tour to take you there and back (from Melbourne).
Task:	Find as MANY pages as you can that describe day tours to the Penguins' Parade from Melbourne.

Figure 6.6: Sample assignments for the user experiment

Two of the twelve assignments defined for the user experiment. For each assignment its title, information need and task were specified.

Next, we constructed document sets for each of the assignments by issuing a MetaCrawler query using the words appearing in the assignment’s title. We downloaded the Web documents corresponding to all the snippets retrieved. We requested 200 documents from each search engine; thus our document sets were between 600 to 1000 documents in size.

Graduate students were asked to complete the twelve assignments using one of two interfaces: a ranked list interface and a dynamic index interface (a Grouper II interface without the merged-clusters interface option). The first four assignments served as training runs for the users and were therefore not considered as part of the experiment. Users performed two assignments on one interface and then were switched to the second interface and so on. The interface they used first was selected at random. The experimental setup was designed such that when a user "followed" a document (i.e., clicked on its title when viewing its snippet), the document was brought from disc, thus avoiding non-uniform network delays.

The users were allowed no more than 5 minutes per assignment. When a user found a document that addressed the information need of the assignment she was performing, she could mark it "relevant" by clicking a special link that was inserted at the top of each document. This action was logged by the system, as were all other user activities.

When analyzing the data we had to determine which documents were actually relevant to each information need. If a majority (but at least two) of the participants viewing a certain document agreed on its relevancy for a specific task, it was automatically marked as such without further investigation. Ties were broken by manual inspection of the documents in question, as were document that only a single person viewed and marked as relevant.

We started by analyzing the assignments in which the task was to find a single document. We wanted to examine whether users of the dynamic index interface had a different success rate or required a different amount of time to complete the assignment compared to users using a ranked list interface. Table 6.1 presents this data. There appears to be little difference in the time the user requires to find one relevant document. Regarding success rate, the ranked list display does appear to have slightly higher success rates.

Table 6.1: Dynamic index vs. ranked list on task requiring one document

	success rate (%)	time to complete (sec.)
ranked list	90%	136.8
dynamic index	78%	133.7

The success rate and the time required to find a relevant document for users of the ranked list interface and the dynamic index interface, given the task of locating a single relevant document. The ranked list interface exhibits slightly higher success rates. There is little difference between the two systems with regards to the time the user requires to find a relevant document.

Table 6.2 compares user performances on assignments in which the task was to find multiple relevant documents. It presents the average number of relevant documents found per session, and the amount of time required to find the first relevant document. The dynamic index interface appears to help the user locate more relevant documents in a given amount of time than the ranked list presentation. The time to find the first relevant document was shorter for users using the ranked list presentation. When comparing this to Table 6.1 it is interesting to note that users of a ranked list presentation were faster at finding the first relevant document on the assignments requiring multiple documents (compared to assignments requiring a single document). This can be explained by the fact that the density of relevant documents was much higher in these assignments (because of the way these were selected). It is interesting to note that user of the dynamic index interface did not appear to be effected by this, and achieved similar performances on the two assignment categories.

Table 6.2: Dynamic index vs. ranked list on task requiring multiple documents

	average number of relevant documents found	**time to find first document (sec.)**
ranked list	7.1	86.6
dynamic index	8.7	130.7

User performance using a ranked list interface compared to a dynamic index interface on assignments in which the task was to find multiple relevant documents. The dynamic index interface appears to help users locate more relevant documents. The time to find the first relevant document is shorter for users using the ranked list presentation.

Figure 6.7 presents the time to locate relevant document beyond the first using a ranked list and a dynamic index interface. From the figure it is apparent that although finding the first relevant document is faster using a ranked list presentation, finding additional relevant documents is faster using the dynamic index interface. These results closely imitate the previously presented results obtained from analyzing the logs the systems (figures 5.8 and 6.5). We believe the inexperience of the users with the Grouper II interface greatly contributes to the time required to find the first relevant document; given additional practice sessions, this time can be significantly cut.

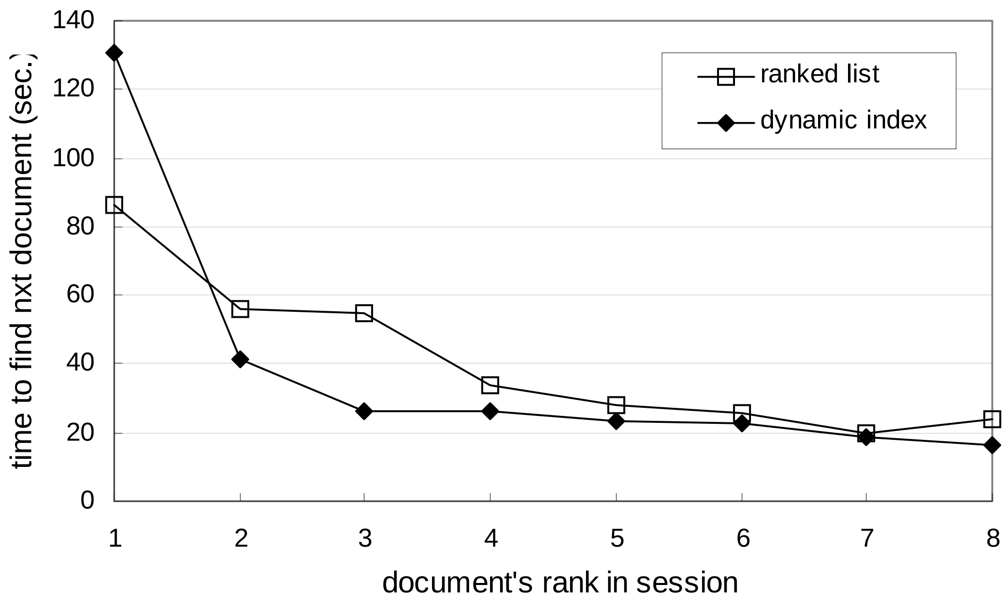


Figure 6.7: Time to locate documents using dynamic index and ranked list

The time to locate relevant documents as a function of the documents' rank in the session using a ranked list and a dynamic index interface. Although finding the first relevant document is faster using a ranked list presentation, finding additional relevant documents is faster using the dynamic index interface.

6.6 Summary

In this chapter we presented the Grouper II system, the second generation of our document clustering interface to a Web search results. Grouper II was designed to address the shortcoming that we have identified in the first Grouper system: the merged clusters are, at times, confusing, not always useful, and do not scale to large document sets.

Grouper II has three basic components that are novel compared to Grouper I. The dynamic index is a new interface paradigm allowing users to view an inverted index of phrases to the retrieved document set. We believe this view is useful in a high portion of the queries (higher than the Grouper I interface) as it does not stumble on meaningless phrases – they are presented but the user can easily avoid them. In Grouper I such phrases are harder to avoid as they might be merged with other phrases (forming a merged cluster). Grouper II also allows the user to view the documents in one of three interfaces: a dynamic index, clusters or a ranked list. And finally, the system supports a hierarchical and interactive interface. Users can define a subset of the retrieved document set. Users then recursively “zoom in” on it. This feature allows users to navigate much larger document sets than the previous “flat” display of Grouper I.

We evaluated the Grouper II system both by analyzing its logs and comparing them to the logs of Grouper I, and by performing a controlled user study comparing the browsing abilities of users given a dynamic index interface and a ranked list interface. Both these evaluations should be viewed as preliminary work because of their scale, the fact that the Grouper II system was not efficiently implemented, and because the users of the system (participants in the user study as well as Web users) were inexperienced with such an interface.

By analyzing the logs of the Grouper II system we were able to show that compared to Grouper I it does a better job at placing together documents that seem interesting to the user. From the user experiment we can tentatively conclude that the dynamic index interface appears to be most useful when the user requires many documents relating to a certain information need. For such tasks Grouper II helps the user find more relevant document (in a given amount of time) and to find them faster. We also suspect that the less frequency the relevant documents are within the document set, the more useful this interface becomes. When faced with the task of finding a single relevant document, the ranked list interface appears to be more appropriate, though we suspect that when the relevant documents are very infrequent the dynamic index interface will actually be better.

Chapter 7

CONCLUSIONS AND FUTURE WORK

7.1 Conclusions

The amount of available information on the Web is increasing rapidly and users rely on search engines to find the information they are looking for. However, finding relevant information using Web search engines often fails. One of the reasons for this is that users typically submit queries that are short and general, retrieving a large numbers of documents, the vast majority of which are of no interest to the user.

The low precision of the Web search engines coupled with the ranked list presentation force users to sift through a large number of documents and make it hard for them to find the information they are looking for. As low precision Web searches are inevitable, tools must be provided to help users "cope" with (and make use of) these large document sets. The motivation for this research was to make search engine results easy to browse, allowing the user to find a relevant documents in the result set even if it is very large and mostly irrelevant.

The central question of this dissertation was: *is the automatic grouping of similar documents (document clustering) a feasible method of presenting the results of Web search engines?* We have identified some key requirements for document clustering of search engine results: similar documents should be clustered together (clustering quality), the system must provide concise and accurate cluster descriptions, and the clustering system should not introduce a substantial delay before displaying the results. In chapter 4 we have carried out extensive offline evaluation of clustering algorithms to assess clustering quality and speed. We have not directly evaluated the descriptive power of our system. Instead, we implemented and deployed a clustering interface to search results on the Web and attempted to evaluate its usefulness to user (chapters 5 and 6).

Another issue investigated in this thesis was the use of phrases for clustering, as phrases contain more information than simple words (information regarding proximity and order of words). Phrases have the equally important advantage of having a higher descriptive power (compared to single words). This is very important when attempting to describe the contents of a cluster. This led us to the second major questions of the dissertation: *will the use of phrases help in achieving high quality groupings of search engine results?*

In this dissertation we presented a novel clustering algorithm – Suffix Tree Clustering (STC) – with two key features: the use of phrases and a simple cluster definition. STC has two main steps. In the first step it searches for all sets of documents that share a common phrase. We call these sets *phrase clusters* and they are found using the suffix tree data structure. In the second step we merge these phrase clusters into clusters (appropriately, we call these *merged clusters*). We merge two phrase clusters based on the percent of the documents that contain both phrases.

Several characteristics make STC a promising candidate for the clustering of search results. First, it is phrase-based, generating clusters by grouping documents that share many phrases. Phrases are also useful in constructing concise and accurate descriptions of the clusters. Second, it does not adhere to any model of the data. Its only assumption is that documents on the same topic will share common phrases. Third, STC allows overlapping clusters. It is important to avoid confining each document to only one cluster since documents often have multiple topics, and thus might be similar to more than one group of documents. Fourth, STC uses a simple cluster definition – all documents containing one of the cluster's phrases are members of the cluster. Finally, STC is a fast incremental linear-time (in the number of documents) algorithm, which makes it suitable for online clustering of Web searches.

We have evaluated the clustering quality of the STC algorithm and compare it to other commonly used (vector-based) clustering algorithms. We used three document collections is our experiment: two Web Search Results (WSR) collections that we constructed for this research (one of search engine snippets and the other of their corresponding documents), and a standard medical collection (OHSUMED) that is used extensively in IR research. The clustering quality was compared using two quality-evaluation approaches: the IR approach and the “merge-then-cluster” approach. The consistency of the results across the different evaluation metrics and text collections was encouraging and added to our confidence in the results. Our experiments demonstrated that STC and k-means produce the highest quality clusters. Of these, STC was superior on the Web collections, while k-means was slightly better on the OHSUMED collection.

We proceeded to investigate the contribution of STC’s features to its success. Was it that STC allows overlapping clusters? We have shown that when overlapping clusters are not allowed, the quality of the STC clusters decreases by more than 20%. We therefore asked could overlapping clusters improve the performance of other clustering algorithms as well? When we adapted the k-means algorithm to allow overlapping clusters we observed a 7% increase in its performance (it was still inferior to STC), suggesting that other clustering algorithms could also benefit from this capability.

STC uses phrases to detect similarity between documents. Would its clustering quality decrease without the use of phrases? Our experiment showed that without the use of multiword phrases, the quality of the clusters created by STC fell by close to 20%.

After showing the advantages of using phrases in STC, we turned our attention to other clustering algorithms. Can phrases improve the performance of vector-based clustering algorithm as well? We used a suffix tree to identify phrases in the document set and added these as attributes to the documents’ vector representations. We were able to show substantial improvement in the performance of all vector-based algorithms tested. The improvement was most noticeable in the algorithms that performed poorly using words alone. We believe this is because using phrases reduces the amount of noise in the data, as chance similarities are less likely to occur.

The STC algorithm identifies phrases using a suffix tree, however alternative phrase generation methods have been explored in the IR literature. Would there be a significant difference in performance between the use of suffix tree phrases compared to other kinds of phrases, say n-grams? We showed that there is only a minor advantage to using suffix tree phrases over using 2-grams. This means that the use of long phrases (3 words or more in length) does not add much to the quality of the results. We believe that the main advantage of using suffix tree phrases lies in their descriptive power – longer phrases are much better at describing the contents of a cluster to the user.

We also wanted to understand why phrases are better than words: Does the advantage of using phrases lie in the information present in the adjacency and order of the words, or simply in their being multiword features? To address these issues we generalized the STC algorithm such that phrase clusters are defined using *frequent sets* – sets of documents that share a set of words. We called this algorithm Frequent Set Clustering (FSC). We were able to show that the quality of FSC clusters is comparable to that of STC on the short documents of WSR-SNIP. However, in the WSR-DOCS collection and even more so in the OHSUMED collection (whose document sets contain many more phrases than WSR-SNIP document sets), STC outperformed FSC. We concluded that the information present in the order and adjacency of words is important and can contribute to the clustering quality.

After investigating the clustering quality of STC, we needed to evaluate its speed. We compared STC to other linear-time clustering algorithms and showed STC to be significantly faster than k-means (the fastest of the vector-based algorithms) when clustering search engine snippets. When clustering Web documents the speed advantage of STC disappears (both STC and k-means are linear with respect to the average document length, but this constant is larger for STC).

In our client-clustering model, clustering of search engine snippets can be performed much faster than clustering their corresponding Web documents. Our clustering algorithm must therefore be *snippet tolerant* – it ought to produce high quality clusters even when it only has access to the snippets returned by the search engines – as most users are unwilling to wait while the system downloads the original documents off the Web. But is snippet clustering a reasonable approximation to document clustering? We compared the performance of clustering algorithms on collections of search engine snippets and on their corresponding Web documents. The quality of STC clusters on snippets was some 20% lower than on Web documents. We believe this to be a moderate decrease in quality that shows that snippet clustering is a reasonable approximation to document clustering. This moderate decrease in quality is surprising as a Web document contained approximately 10 times more words on average than a snippet. One explanation is that the snippets represent (partially successful) attempts by the search engines to extract meaningful words and phrases from the original documents.

After evaluating the STC algorithm and showing it to be fast and to produce high quality clusters we turned our attention to evaluating the clustering of search results from the users' perspective – Will such a system be in fact useful to the users? And if so, for what information tasks is it most suited? In order to evaluate these questions we have implemented and deployed a clustering interface to Web searches. This allowed us to obtain quantitative evaluations via analysis of the system's logs and a user study as well as qualitative impressions via user feedback.

Grouper is, to our knowledge, the first implementation of a post-retrieval document-clustering interface to a Web search engine. Grouper uses the HuskySearch meta search engine to retrieve results from several popular Web search engines, then clusters the results using the STC algorithm. Grouper is publicly available at http://www.cs.washington.edu/research/clustering.

Grouper I presents the merged clusters of the STC algorithm to the user. It present the merged clusters ordered by their estimated quality. The phrases (of the phrase clusters) that created the cluster are used to describe its content to the user.

By fielding Grouper I on the Web, we were able to identify some of its shortcomings. Sometimes, Grouper I identifies phrases that are common in the collection but are of no interest to the user. When they are used to construct clusters the result might be random and useless clusters. Even the merging of meaningful phrases can at times create confusing clusters. We have also noticed that the sizes and/or the number of "useful" clusters created by STC increases as more documents are retrieved. This means that for large document sets the users still have to scan many documents either because they scan many clusters or because the clusters they scan are large. As we are interested in making the system scale to substantially larger retrieved document sets, clusters should be presented hierarchically so users can navigate the results more efficiently.

Grouper II was designed to address these shortcomings. Grouper II has three basic components that are novel compared to Grouper I:

1. **Dynamic Index:** This interface presents non-merged phrase clusters. This view is essentially an inverted index of phrases to the retrieved document set. It is useful in a higher portion of the queries (compared to the Grouper I interface), as users can easily ignore meaningless phrases and there are no mergers of phrase clusters that might be erroneous. The dynamic index is generated by selecting a subset of the phrases identified by the suffix tree.
2. **Multiple Views:** The system allows the user to view the documents in one of three interfaces: a dynamic index, clusters or a ranked list.
3. **Interactive and Hierarchical Navigation:** Grouper II supports a hierarchical and interactive interface, similar to the Scatter/Gather interface [Cutting et. al., 92]. Users can select one or more index entries or clusters to focus on, thus defining a subset of the retrieved document set. Users then are able to recursively "zoom in" on this subset (viewing it via one of the three possible views). This feature allows users to navigate much larger document sets than the previous "flat" display of Grouper I.

Is Grouper useful? How does it compare a ranked list interface (such a HuskySearch)? In what cases is it better? We addressed these questions in two ways. First, we utilized the logs of the deployed Grouper system as the basis of our evaluation. We compared these logs to the logs of the HuskySearch system (which is identical except for having a ranked list interface). Second, we performed a controlled user study using the Grouper II interface giving users specific tasks and specific retrieved document sets (thus evaluating purely their ability to browse the retrieved documents).

Analyzing the logs of the Grouper I system we were able to find significant differences between the behaviors of Grouper I users and of HuskySearch users. First, Grouper user had fewer “dead-end sessions”. A dead-end session is a session in which the user does not follow even a single document (typically because she could not find anything of interest in the result set). Second, we observed that finding the first few interesting documents actually requires more “effort” in Grouper as compared to HuskySearch’s ranked-list presentation, but after the first two or three documents, finding additional interesting documents appears to require less effort in Grouper. This possibly reflects the fact that the user must spend some time/effort to understand the clusters that Grouper presents, but after doing so the clusters are helpful in finding required information faster. It also confirms our observation that a clustering interface is not suitable for all search tasks. By analyzing the logs of the Grouper II system we were able to show that compared to Grouper I it does a better job at placing together documents that seem interesting to the user.

Our small-scale user experiment compared the dynamic index interface of Grouper II to a ranked list interface on a set of eight predetermined queries and the documents they retrieved. From this experiment we tentatively concluded that the dynamic index interface appeared to be most useful when the user required many documents relating to a certain information need. For such tasks Grouper II helped the user find more relevant documents (in a given amount of time) and to find them faster. We believe that the less frequent the relevant documents are within the document set, the more useful this interface becomes. When faced with the task of finding a single relevant document, the ranked list interface appeared to be more appropriate, though we suspect that when the relevant documents are very infrequent the dynamic index interface will actually be better.

7.2 Future Work

In this section we present some directions for future work.

7.2.1 Implementation Issues

As stated previously, Grouper II was implemented towards the end of this study; its implementation was "quick and dirty" resulting in slower performance than could have been achieved with additional engineering efforts. For example, each user iteration is a separate process and all communications between these processes is done via files on disk. Additional improvements will reduce the number of times the data is copied as well as speed up the parsing subroutines. We believe that these improvements can cut the clustering time by some 50%.

Following we detail the resources needed to run the Grouper system on a commercial site. The CPU time (on a Pentium 200Mhz processor) and the memory requirements (per query) are presented in table 7.1. These are our best estimates combining our empirical data with estimates regarding the speed-ups that would be achieved by implementation enhancements. These numbers relate to the first iteration of the user (i.e., when submitting the query). Subsequent iteration are much faster as the documents do not need to be parsed again, the suffix tree does not need to be built and the number of phrase clusters to consider is typically much smaller.

Table 7.1: The CPU time and memory needed by Grouper

number of snippets	100	200	300	400	500
parsing time (sec.)	0.22	0.44	0.66	0.88	1.10
suffix tree construction time (sec.)	0.20	0.40	0.60	0.80	1.0
score and sort phrase clusters time (sec.)	0.24	0.27	0.29	0.32	0.35
dynamic index selection time (sec.)	0.05	0.05	0.05	0.05	0.05
merged clusters generation time (sec.)	0.53	0.53	0.53	0.53	0.53
total dynamic index time (sec.)	0.71	1.16	1.60	2.05	2.55
total merged clusters time (sec.)	0.21	1.66	2.10	2.55	3.05
required memory (KB)	250	500	750	1000	1250

The CPU time (on a Pentium 200Mhz processor) and the memory required by the Grouper system, per query. These are our best estimates combining our empirical data with estimates regarding the speed-ups that would be achieved by implementation enhancements. These numbers relate to the first iteration of the user (i.e., when submitting the query) – subsequent iteration are much faster.

Grouper is implemented using a single process. To make the iterative session fast, the process' data-structures have to remain "alive" while the user is zooming in (this avoids the need to save the data to disk, to re-parse it and to rebuild the suffix tree). This means the process has to remain running or the data-structures have to be saved (*e.g.*, in an object database).

7.2.2 Statistical Analysis of Phrases in Documents

We need a better understanding of the statistics of phrase appearance in documents. In this work we have presented statistics regarding the number of phrases of different lengths the suffix tree has found in Web documents, search engine snippets and OHSUMED abstracts. But there are additional question to be asked: What is the frequency distribution of phrases of different lengths in different text collection? How does this distribution differ between different kinds of phrases (and multiword features) such as n-grams, suffix tree phrases and frequent sets? How often do words that appear frequently in a phrase also appear together, but not adjacent, in a document, and how often do they appear by themselves? Answers to these questions might give us better understanding how to assess a phrase's importance. They might also help us weight phrases more wisely when using them as features in the vector representations of documents.

7.2.3 Effect of Snippet Generation Technique on Clustering Quality

In this research the search engine snippets were gathered by HuskySearch, which collates snippets from a number of different search engines. We used collated sets as they are representative of current snippet technology (our WSR-SNIP snippet collection is representative of the time it was created – November 1997). Search engines vary greatly in the techniques used to generate snippets. A Snippet can be static, appearing the same every time the page is retrieved, or it can be dynamic, sensitive to the query that retrieved the page. Static snippets can be generated using advance summarization methods, or they can simply be the first 200 Bytes of text on the page. Dynamic snippets can also be generated in various forms. Some engines generate snippets solely from the text of the page, while others incorporate the text of the links pointing at the page as well.

We hypothesize that the quality of the clustering is highly effected by the techniques used to create the snippets. More sophisticated techniques (*e.g.*, dynamic snippets, using text on links) might cause more phrases to be found in the result set. We also hypothesize that, on average, today's snippets are better than the snippets we collected in 1997, and therefore the results of clustering today's snippets would be of higher quality than achieved in this research.

7.2.4 Investigating Phrase using Modified Variations of Frequent Sets

Phrases are ordered, contiguous multiword features, whereas frequent sets are non-ordered non-contiguous multiword features. In this dissertation we have investigated the advantages of using phrases over single words by comparing phrases to frequent sets. This allowed us to conclude that the information present in the order and adjacency of words is important and does contribute to the clustering quality.

We can further explore this issue by investigating variations of FSC that use ordered non-contiguous multiword features or non-ordered contiguous features. It would also be interesting to explore the benefits of words that are non-contiguous but near to each other (say within 5 words of each other) and to limit word set sizes. These alternatives can be thought of as “approximate phrases”, and evaluating their contribution to the clustering quality can help us further understand the reason phrases are better than words.

7.2.5 Incorporating Relevancy Information in STC

The STC algorithm, like all clustering algorithms, treats all documents equally. But documents are not equal – they have different relevance to the query. The relevance of the documents to the query diminishes as more documents are retrieved. For some queries (*e.g.*, very general one), the decrease in relevancy is very moderate; for other queries it can be quite significant. Producing clusters based on many documents of low relevance might degrade the quality of the results.

One possible solution to this problem is to have the clustering algorithm automatically decide which documents to exclude from the cluster generation process (they may be added to the clusters once these are created). Another approach is to modify the STC algorithm so that it takes into consideration the relevancy of each document. There are two components of the algorithm that should be modified. First, the phrase cluster scoring function should be modified to take into account not only the number of documents a phrase appears in but also the relevancy of those documents.

Second, when determining whether two phrase clusters should be merged based on the percentage of documents that contain both phrases, all documents should not be treated as equal – their relevance to the query should be taken into account.

7.2.6 Clustering a Sample of the Documents

In many clustering problems, when the domain is large clustering is performed on a sample of the items in the domain [*e.g.*, Guha et al., 98]. Grouper currently clusters all the documents that are returned by the search engine. Would the clustering quality decrease dramatically if Grouper generated its clusters based on a sample of the documents? If not – sampling might be a useful technique when scaling the system to larger document sets.

7.2.7 Clustering on an Indexing Search Engine

Our model for this work has been to separate the clustering module from the search engine. A different model is to perform the clustering on the search engine itself. This entails severe resource limitations on the clustering module at query time. However, the documents can be preprocessed for clustering at indexing time. Can we use STC to cluster search results in such a scenario? Can we create a dynamic index?

One possible way this can be done is to create (at index time) an inverted index file not only of words but also of 2-grams, and perhaps even of 3-grams. We use this inverted index to identify common phrases (phrases that appear in more than a certain number of documents) and disregard the uncommon ones. For each document we keep a list of all the common phrases it contains. At query time we collect the common phrases of all the retrieved documents, and count the number of documents each phrase appears in. These can be used as phrase clusters.

Such a design might increase the indexing time of a collection by as much as 100% as well as increase the amount of memory needed. On the other hand, the phrase clusters can be generated with very little query time processing. These phrase clusters can be used as input to the third step of the STC algorithm (which might require some time, as it is quadratic in the number of phrase clusters used), or for the generation of a dynamic index (which is very fast). This approach will actually allow more sophisticated processing of the phrases as more information will be available – the frequency of each phrase in the complete corpus will be known, not only its frequency in the results set (the same is true for the statistics on phrase co-occurrence well).

7.2.8 Improving the User Interface

The current user interface of Grouper, whether clusters or dynamic index, is very rudimentary. There is additional information that can be presented to the user and might be of value. For example, it might be interesting for the user to know how similar the documents of each cluster are and how relevant they are to the query. The similarities between clusters might also be of interest – clusters can be organized not alphabetically but in a 2-dimentional Self-Organizing Map with distances between clusters reflecting the similarities between their document groups.

The current interface is implemented solely in HTML. What can be done if we use Java or a dedicated client? For example, instead of having a fixed threshold that determines when phrase clusters should be merged in STC, the user can have a sliding bar determining this threshold and view the clusters change as the threshold is changed. Different thresholds could even be applied to different clusters.

7.2.9 User Studies

The controlled user study performed in this thesis was of small scale. It focussed on the browsing ability of the dynamic index interface compared to a ranked list interface. A larger user study (including 40-100 participants) would allow drawing more statistically significant conclusions. The participants in the study must be given sufficient experienced with the Grouper II interface before the study commences. The clustering interface should also be evaluated in a controlled user study, as well as Grouper's usefulness in query reformulation. We had hypothesized that Grouper would be more useful when the density of relevant documents in the result set is low. This can also be studied via a user study.

7.2.10 The Use of Frequent Sets in Additional Information Retrieval Tasks

Finally, we have introduced in this thesis the possibility of using frequent sets as multiword features in IR. This idea seems especially appropriate for the task of document classification. Frequent sets should be identified in each class of document and should then be used as possible features for the classification algorithm.

BIBLIOGRAPHY

[Agrawal et al., 93] Agrawal, R., Imielinski, T. and Swami, A. Mining associations between sets of items in massive databases. In *Proceedings of the ACM-SIGMOD 1993 International Conference on Management of Data* (SIGMOD'93), 207-216, 1993.

[Agrawal and Srikant, 94] Agrawal, R. and Srikant, R. Fast algorithms for mining association rules. In *Proceedings of the 20th International Conference on Very Large Databases* (VLDB'94), 1994.

[Allan et al., 96]. Allan, J., Ballesteros, L., Callan, J. P., Croft, W. B. and Lu, Z. Recent experiments with INQUERY. In D. K. Harman (ed.), *The Fourth Text Retrieval Conference* (TREC-4), NIST Special Publication, 1996.

[Allen et al., 93] Allen, R. B., Obry, P. and Littman, M., An interface for navigating clustered document sets returned by queries. In *Proceedings of the ACM Conference on Organizational Computing Systems* (COOCS'93), 166-171, 1993.

[Bayardo, 97] Bayardo, R. J. Brute-force mining of high-confidence classification rules. In *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining* (KDD'97), 123-126, 1997.

[Bayardo, 98] Bayardo, R. J. Efficiently mining long patterns from databases. In *Proceedings of ACM SIGMOD International Conference on Management of Data* (SIGMOD'98), 85–93, 1998.

[Boyan et al., 96] Boyan, J., Freitag, D. and Joachims, T. A machine architecture for optimizing Web search engines. In *Proceedings of the AAAI-96 Workshop on Internet based Information Systems*, 1996.

[Brin et al., 97] Brin, S., Motwani, R., Ullman, J. and Tsur, S. Dynamic itemset counting and implication rules for market basket data. In *Proceedings of ACM SIGMOD International Conference on Management of Data* (SIGMOD’97), 255-264, 1997.

[Brin and Page, 98] Brin, S. and Page, L. The anatomy of a large-scale hypertextual Web search engine. In *Proceedings of the Seventh International Web Wide World Conference* (WWW7), 1998.

[Broder et al., 97] Broder, A. Z., Glassman, S. C., Manasse, M. S. and Zweig, G. Syntactic clustering of the Web. In *Proceedings of the Sixth International Web Wide World Conference* (WWW6), 1997.

[Brown et al., 90] Brown, P. F., Cock, J., Della Pietra, S. A., Della Pietra, V. J., Jelinek, F., Lafferty, J. D., Mercer, R. L. and Roossin, P. S. A statistical approach to machine translation. *Computational Linguistics*, 16(2):79-85, 1990.

[Card et al., 91] Card, S. K., Robertson, G. G. and Mackinlay, J. D. The information visualizer: An information workspace. In *Proceedings of ACM Human Factors in Computing Systems Conference* (CHI'91), 181-188, 1991.

[Chalmers & Chiton, 92] Chalmers, M. and Chitson, P. BEAD: Explorations in information visualization. In *Proceedings of the 15th International ACM SIGIR Conference on Research and Development in Information Retrieval* (SIGIR'92), 330-337, 1992.

[Chalmers, 96] Chalmers, M. A linear iteration time layout algorithm for visualizing high-dimensional data. In *Proceedings of Visualization*, 127-132. 1996.

[Cheesman et al., 88] Cheeseman, P., Kelly, J., Self, M., Stutz, J., Taylor, W. and Freeman, D. AutoClass: A bayesian classification system. In *Proceedings of the Fifth International Conference on Machine Learning* (ML’88), 54-64, 1988.

[Cheeseman and Stutz, 96] Cheeseman, P. and Stutz, J. Bayesian classification (AutoClass): Theory and results. In Fayyad, U. M., Piatetsky-Shapiro, G., Smyth, P. and Uthurusamy, R. (eds.) *Advances in Knowledge Discovery and Data Mining*, AAAI Press/MIT Press, 1996.

[Cooper, 88] Cooper, W. S., Getting beyond Boole. *Information Processing and Management*, 24(3):243-248, 1988.

[Croft, 78] Croft, W. B. Organizing and searching large files of documents. Ph.D. Thesis, University of Cambridge, 1978.

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

[Cutting et al., 93] Cutting, D. R., Karger, D. R and Pedersen, J. O. Constant interaction-time Scatter/Gather browsing of large document collections. In *Proceedings of the 16th International ACM SIGIR Conference on Research and Development in Information Retrieval* (SIGIR'93), 126-135, 1993.

[Dillon and Grey, 83] Dillon, M. and Gray, A. FASIT: A fully automatic syntactically based indexing system. *Journal of the American Society for Information Science*, 34(2):99-108, 1983.

[Dumais and Zamir, 99] Dumais, S. and Zamir, O. Unpublished work, 1999.

[Dunlop, 97] Dunlop, M. D. Time, relevance and interaction modeling for information retrieval. In *Proceedings of the 20th International ACM SIGIR Conference on Research and Development in Information Retrieval* (SIGIR'97), 206-215, 1997.

[Ehrenfeucht and Haussler, 86] Ehrenfeucht, A. and Haussler, D. A new distance metric on strings computable in linear time. *Discrete Applied Math*, 40, 1988.

[Fagan, 87] Fagan, J. L. Experiments in automatic phrase indexing for document retrieval: A comparison of syntactic and non-syntactic methods. Ph.D. Thesis, Cornell University, 1987.

[Feldman et al., 97] Feldman, R., Aumann, Y., Amir, A., Kloesgen, W. and Zilberstien, A. Visualization techniques to explore data mining results for document collections. In *Proceedings of the 3rd International Conference on Knowledge Discovery* (KDD'97), 16-25, 1997.

[Fowler et al., 91] Fowler, R. H., Fowler, W. A. and Wilson, B. A., Integrating query, thesaurus, and documents through a common visual representation. In *Proceedings of the 14th International ACM SIGIR Conference on Research and Development in Information Retrieval* (SIGIR'91), 142-151, 1991.

[Fowler et al., 95] Fowler, R. H., Fowler, A. L., Kumar, A. and Williams, J. L. Head-couples stereo display for visualization in a document retrieval system using associated networks. System demonstration abstract in *Proceedings of the 18th International ACM SIGIR Conference on Research and Development in Information Retrieval* (SIGIR'95), 1995.

[Furnkranz, 98] Furnkranz, J. A study using n-gram features for text categorization. Technical Report OEFAI-TR-98-30, Austrian Research Institute for Artificial Intelligence, 1998.

[Furnkranz et al., 98] Furnkranz, J., Mitchell, T. and Riloff, E. A case study in using linguistic phrases for text categorization on the WWW. In *Proceedings of 1998 AAAI/ICML Workshop on Learning for Text Categorization*, 1998.

[Griffiths et al., 86] Griffiths, A., Luckhurst, H. C. and Willet, P. Using inter-document similarity information in document retrieval systems. *Journal of the American Society for Information Science,* 37:3-11, 1986.

[Guha et al., 98] Guha, S., Rastogi, R. and Shim, K. CURE: An efficient clustering algorithm for large databases. In *Proceedings of the ACM-SIGMOD 1998 International Conference on Management of Data* (SIGMOD’98), 73-84, 1998.

[Gusfield, 97] Gusfield, D. *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*, chapter 6, Cambridge University Press, 1997.

[Harter, 96] Harter, S. P. Variations in relevance assessments and the measurement of retrieval effectiveness. *Journal of the American Society for Information Science,* 47(1):37-49, 1996.

[Hearst, 94] Hearst, M. A. Using categories to provide context for full-text retrieval results. In *Proceedings of the 1994 RIAO Conference* (RIAO'94), 1994.

[Hearst, 95] Hearst, M. A. TileBars: Visualization of term distribution information in full text information access. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems* (CHI'95), 59-66, 1995.

[Hearst and Pederson, 96] Hearst, M. and Pedersen, P. Reexamining the cluster hypothesis: Scatter/gather on retrieval results. In *Proceedings of the 19th International ACM SIGIR Conference on Research and Development in Information Retrieval* (SIGIR'96), 76-84, 1996.

[Hearst and Karadi, 97] Hearst, M. and Karadi, C. Cat-a-Cone: An interface for specifying searches and viewing retrieval results using a large category hierarchy. In *Proceedings of the 20th International ACM SIGIR Conference on Research and Development in Information Retrieval* (SIGIR'97), 1997.

[Hearst, 98] Hearst, M. A. The use of categories and clusters in information access interfaces. In T. Strzalkowski (ed.), *Natural Language Information Retrieval*, Kluwer Academic Publishers, 1998.

[Hemmje et al., 94] Hemmje, M., Kunkel, C. and Willet, A. LyberWorld: A visualization user interface supporting fulltext retrieval. In *Proceedings of the 17th International ACM SIGIR Conference on Research and Development in Information Retrieval* (SIGIR'94), 249-259, 1994.

[Hersh et al., 94] Hersh, W. R., Buckley, C., Leone, T. J. and Hickam, D. H. OHSUMED: An interactive retrieval evaluation and new large test collection for research. In *Proceedings of the 17th International ACM SIGIR Conference on Research and Development in Information Retrieval* (SIGIR'94), 192-201, 1994.

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

[Hull, 96] Hull, D. A. Stemming algorithms: A case study for detailed evaluation. *Journal of the American Society for Information Science*, 47(1):70-84, 1996.

[Hull et al., 97] Hull, D. A., Grefenstette, G., Schulze, B. M., Gaussier, E., Schütze, H. and Pedersen, L. O. Xerox TREC-5 site report: Routing, filtering, NLP, and Spanish tracks. In: D. K. Harman (ed.), *The Fifth Text Retrieval Conference (TREC-5)*. NIST Special Publication, 1997.

[Jardine and van Rijsbergen, 71] Jardine, N. and van Rijsbergen, C. J. The use of hierarchical clustering in information retrieval. *Information Storage and Retrieval*, 7:217-240, 1971.

[Kehoe and Pitkow, 99] Kehoe, C. and Pitkow, J. Tenth WWW Survey Report, The Graphics, Visualization & Usability Center, Georgia Institute of Technology, 1999.

[Kim & Korfhage, 94] Kim, H. and Korfhage, R. R. BIRD: Browsing Interface for the Retrieval of Documents. In *Proceedings of the IEEE Symposium on Visual Languages*, 176-177, 1994.

[Kleinberg, 98] Klienberg, J. M. Authoritative sources in a hyperlinked environment. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, 1998.

[Konhonen, 90] Kohonen, T. The self-organizing map. *Proceedings of the IEEE*, 78(9):1464-1480, 1990.

[Konhonen, 97] Kohonen, T. Exploration of very large databases by self-organizing maps. In *Proceedings of the IEEE International Conference on Neural Networks*, 1, 1-6, 1997.

[Korfhage, 91] Korfhage, R. To see, or not to see - is that the query? In *Proceedings of the 14th International ACM SIGIR Conference on Research and Development in Information Retrieval* (SIGIR'91), 134-141, 1991.

[Krellenstein, 98] Krellenstein, M., Chief Technology Officer, Northern Light Technology LLC. The added value of classification intelligence. Talk at the 1998 Search Engines and Beyond Conference, 1998.

[Landau and Vishkin, 89] Landau, G. M. and Vishkin, U. Fast Parallel and serial approximate string matching. *Journal of Algorithms*, 10, 157-169, 1989.

[Lawrence and Giles, 99] Lawrence, S. and Giles, L. Accessibility and Distribution of Information on the Web. *Nature*, 400, 107-109, July 1999.

[Leouski & Croft, 96] Leouski, A., and Croft, W. B. An evaluation of techniques for clustering search results. Technical report IR-76, Department of Computer Science, University of Massachusetts, Amherst, 1996.

[Lewis, 92] Lewis, D. D. An evaluation of phrasal and clustered representation on a text categorization problem. In *Proceedings of the 15th International ACM SIGIR Conference on Research and Development in Information Retrieval* (SIGIR'92), 37-50, 1992.

[Lin, 91] Lin, X. A self-organizing semantic map for information retrieval. In *Proceedings of the 14th International ACM SIGIR Conference on Research and Development in Information Retrieval* (SIGIR'91), 262-269, 1991.

[Lin, 93] Lin, X. Self-organizing semantic maps as graphical interfaces for information retrieval. Ph.D. Thesis, University of Maryland, 1993.

[Lin, 97] Lin, X. Map displays for information retrieval. *Journal of the American Society for Information Science*, 48(1): 40-54, 1997.

[Lin and Kedem, 98] Lin, D. and Kedem, Z. M. Pincer-Search: A new algorithm for discovering the maximum frequency set. In *Proceedings of the Sixth European Conference on Extending Database Technology*, 1998.

[Maarek and Wecker, 94] Maarek, Y. S. and Wecker, A. J. The Librarian's Assistant: Automatically organizing on-line books into dynamic bookshelves. In *Proceedings of the 1994 RIAO Conference* (RIAO'94), 1994.

[Macskassy et al., 98] Macskassy, S., Banerjee, A., Davison, B. and Hirsh H. Human performance on clustering Web pages: A preliminary study. In *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining* (KDD’98), 1998.

[Manber and Myers, 93] Manber, U. and Myers, G. Suffix arrays: A new method for on-line search. *SIAM Journal of Computation*, 22:935-948, 1993.

[McCreight, 76] McCreight, E. M. A space-economical suffix tree construction algorithm. *Journal of the ACM*, 23:262-272, 1976.

[Merkl, 97] Merkl, D. Exploration of text collections with hierarchical feature maps. In *Proceedings of the 20th International ACM SIGIR Conference on Research and Development in Information Retrieval* (SIGIR'97), 186-195, 1997.

[Milligan et al., 83] Milligan, G. W., Soon, S. C. and Sokol, L. M. The effect of cluster size, dimensionality and the number of clusters on recovery of true cluster structure. *IEEE Transactions on Pattern Analysis and Machine Intelligence.* vol. PAMI-5, no. 1, 1983.

[Milligan and Cooper, 85] Milligan, G. W. and Cooper, M. C. An examination of procedures for detecting the number of clusters in a data set. *Psychometrika*, 50:159-179, 1985.

[Mladenic and Grobelnik, 98] Mladenic, D. and Grobelnik, M. Word sequences as features in text learning. In *Proceesdings of the 17th Electrotechnical and Computer Science Conference* (ERK'98), 1998.

[Motro, 98] Motro, H., Infoseek CEO, CNBC, May 7, 1998.

[Mukherjea et al., 95] Mukherjea, S., Foley, J. D. and Hudson, S. Visualizing complex hypermedia networks through multiple hierarchical views. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems* (CHI'95), 331-337, 1995.

[Nowell et, al., 96] Nowell, L. T., France, R. K., Hix, D., Heath, L. S. and Fox, E. Visualizing search results: Some alternatives to query document similarity. In *Proceedings of the 19th International ACM SIGIR Conference on Research and Development in Information Retrieval* (SIGIR'96), 67-75, 1996.

[Nuchprayoon & Korfhage, 97] Nuchprayoon, A. and Korfhage, R. R. GUIDO: Visualizing document retrieval. In *Proceedings of the IEEE Information Visualization symposium*, 184-188, 1997.

[Olsen et al., 93] Olsen, K. A., Korfhage, R. R., Spring, M. B., Sochats, K. M. and Williams, J. G. Visualization of a document collection: The VIBE system. *Information Processing and Management*, 29(1):69-81, 1993.

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

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

[Rasmussen, 92] Rasmussen, E. Clustering Algorithms. In Frakes, W. B. and Baeza-Yates, R. (eds.), *Information Retrieval*, Prentice Hall, Eaglewood Cliffs, N. J., 419-442, 1992.

[Rocchio, 66] Rocchio, J. J. Document retrieval systems - optimization and evaluation. Ph.D. Thesis, Harvard University, 1966.

[Rodeh et al., 81] Rodeh, M., Pratt, V. R. and Even, S. Linear algorithm for data compression via string matching. *Journal of the ACM*, 28(1):16-24, 1981.

[Rose and Belew, 91] Rose, D. E. and Belew, R. K. Toward a direct-manipulation interface for conceptual information retrieval systems. In Martin Dillon (ed.), *Interfaces for information retrieval and online systems*, Greenwood Press, 39-54, 1991.

[Salton, 71] Salton, G. Cluster search strategies and the optimization of retrieval effectiveness. In Salton, G. (ed), *The SMART Retrieval System*, Prentice-Hall, Englewood Cliffs, N.J., 223-242, 1971.

[Salton et al., 75] Salton, G., Yang, C. S. and Yu, C. T. A theory of term importance in automatic text analysis. *Journal of the American Society for Information Science*, 26(1):33-44, 1975.

[Salton and Buckley, 88] Salton, G., and Buckley, C. Term-weighting approaches in automatic text retrieval. *Information Processing and Management*, 24(5):513-523, 1988.

[Salton, 89] Salton, G. *Automatic Text Processing*. Addison-Wesley, New York, 1989.

[Schütze and Silverstein, 97] Schütze, H. and Silverstein, C. Projections for efficient document clustering. In *Proceedings of the 20th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 74-81, 1997.

[Selberg and Etzioni, 95] Selberg, E. and Etzioni O. Multi-service search and comparison using the MetaCrawler. In *Proceedings of the 4th World Wide Web Conference* (WWW4), 1995.

[Silverstein & Pederson, 97] Silverstein, C. and Pedersen, J. O. Almost-constant time clustering of arbitrary corpus subsets. In *Proceedings of the 20th International ACM SIGIR Conference on Research and Development in Information Retrieval* (SIGIR'97), 60-66, 1997.

[Spoerri, 93] Spoerri, A. InfoCrystal: A visual tool for information retrieval and management. In *Proceedings of Information Knowledge and Management* (CIKM'93), 150-157, 1993.

[Strzalkowski et al., 97] Strzalkowski T. et al.. Natural language information retrieval: TREC-5 report. In: D. K. Harman (ed.), The Fifth Text Retrieval Conference (TREC-5). NIST Special Publication, 1997 ZZZ

[Swan and Allan, 98] Swan, R. and Allan, J. Aspect Windows, 3-D visualizations, and indirect comparisons of information retrieval system. In *Proceedings of the 21th International ACM SIGIR Conference on Research and Development in Information Retrieval* (SIGIR'98), 1998.

[TDT, 97] *Proceedings of the Topic Detection and Tracking (TDT) Workshop*, University of Maryland, College Park, MD, October 1997. ZZZ

[Thompson and Croft, 89] Thompson, R. H. and Croft, W. B. Support for browsing in an intelligent text retrieval system. *International Journal of Man-Machine Studies*, 30(6):639-668, 1989.

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

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

[Veerasamy & Blekin, 96] Veerasamy, A. and Belkin, N. J. Evaluation of a tool for visualization of information retrieval results. In *Proceedings of the 19th International ACM SIGIR Conference on Research and Development in Information Retrieval* (SIGIR'96), 85-92, 1996.

[Veerasamy & Heikes, 97] Veerasamy, A. and Heikes, R. Effectiveness of a graphical display of retrieval results. In *Proceedings of the 20th International ACM SIGIR Conference on Research and Development in Information Retrieval* (SIGIR'97), 1997.

[Voorhees, 85] Voorhees, E. M. Effectiveness and efficiency of agglomerative hierarchic clustering in document retrieval. PhD thesis, Cornell University, 1985.

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

[Wiener, 73] Weiner, P. Linear pattern matching algorithms. In *Proceedings of the 14th Annual Symposium on Foundations of Computer Science* (FOCS’73), 1-11, 1973.

[Willet, 88] P. Willet. Recent trends in hierarchical document clustering: A critical review. *Information Processing and Management*, 24:577-597, 1988.

[Zaki et al., 97] Zali, M. J., Parthasarathy, S., Ogihara, M. and Li, W. New algorithms for fast discovery of association rules. In *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining* (KDD'97), 283-286, 1997.

[Zamir et al., 97] Zamir, O., Etzioni, O., Madani O. and Karp, R. M. Fast and intuitive clustering of Web documents. In *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining* (KDD'97), 287-290, 1997.

[Zamir and Etzioni, 98] Zamir, O. and Etzioni, O. Web document clustering: A feasibility demonstration. In: *Proceedings of the 19th International ACM SIGIR Conference on Research and Development in Information Retrieval* (SIGIR'98), 46-54, 1998.

[Zhai et al., 97] Zhai, C., Tong, X., Milic-Frayling N. and Evans, D. A. Evaluation of syntactic phrase indexing - CLARIT NLP track report. In: D. K. Harman (ed.), The Fifth Text Retrieval Conference (TREC-5). NIST Special Publication, 1997.

VITA

Oren Zamir

U.S. address: CS Department, University of Washington, Box 352350,
Seattle, WA 98195-2350, USA
Israel address: 22 Hatana'im st. apt. 23, Ramat-Aviv, Tel-Aviv 69209, Israel;
zamir@cs.washington.edu;
http://www.cs.washington.edu/homes/zamir

Education:

1995-1999 Ph.D. in Computer Science at the University of Washington, Seattle.
Areas of interest: Information Retrieval, User Interfaces, Machine Learning and Knowledge Discovery.
M.S. received 1997.

Thesis: Clustering Web Documents - A Phrase-Based Method for Grouping Search Engine Results.
The motivation for this research was to make it easier to find relevant information using Web search engines. Document clustering algorithms group similar documents together. Clustering Web search engine results provided the user with a powerful browsing tool that can help find document of interest faster, and find more such documents. This work included the development of novel clustering algorithms and the implementation and deployment of a working system on the Web.

1986-1989 B.Sc. in Physics and Mathematics at the Hebrew University, Jerusalem (as a member of the Talpiyot project). Graduation Cum Laude.

Professional Experience:

1997-1998 Part-time consultant with two VC companies: D.S.Polaris and IsraTech. Evaluated Internet and Information Retrieval startup companies to assess their technical capabilities, development risks, development time and costs, and barriers to entry.

1994-1995 Senior software team leader, Aurec Inc. Worked on the development of a large customer care and billing system for the cellular communications industry. The product included an Oracle database, powerbuilder clients and a Tuxedo transaction management system. Responsibilities: software design, test design and technology assessment.

1989-1994 R&D Officer in the Israeli Defence Forces as a graduate of the "Talpiyot" project.

1993-1994 Project Manager, IDF Military Intelligence. In charge of the analysis and the definition of software requirements for a large software development project (>300 man years). The project involved a large state-of-the-art document management and retrieval system that was designed with novel advanced features. The project included the development of smart document-handling algorithms. Collaborated with researchers at the Computer Science Department at Bar-Ilan University.

1992-1993 Instructor for future I.D.F. R&D officers in the Talpiyot project. In charge of the training and the academic education of 25 cadets in their first year of the Talpiyot project.

1990-1992 Software team leader. Headed the software development team of a real-time, DSP-based signal monitoring and analysis system. The team included 4 programmers and electrical engineers.

1989-1990 Software developer. Developed simulations of communications systems and an expert system for resource allocation. The projects were done at the Israeli Aircraft Industries - the Mabat plant.

Selected Publications:

1999 Clustering Web Documents - A Phrase-Based Method for Grouping Search Engine Results. PhD Thesis, Department of Computer Science and

Engineering, University of Washington.

Grouper: A Dynamic Clustering Interface to Web Search Results, Oren Zamir and Oren Etzioni, in Proceedings of the 8th International World Wide Web Conference.

1998 Web Document Clustering: A Feasibility Demonstration, Oren Zamir and Oren Etzioni, in Proceedings of the 21st Int. SIGIR Conference in Information Retrieval.

TextVis: An Integrated Visual Environment for Text Mining, David Landau, Ronen Feldman, Yonatan Aumann, Moshe Fresko, Yehuda Lindell, Orly Lipshtat and Oren Zamir, in Proc. of the 2nd European Symposium on Principles of Data Mining and Knowledge Discovery.

Visualization of Search Results in Document Retrieval Systems, Oren Zamir, General Examination's Paper, University of Washington.

1997 Fast and Intuitive Clustering of Web Document, Oren Zamir, Oren Etzioni, Omid Madani and Richard Karp, in Proceeding of the 3rd International Conference on Knowledge Discovery and Data Mining.