INFORMATION FORAGING THROUGH CLUSTERING AND SUMMARIZATION: A SELF-ORGANIZING APPROACH

by

Dmitri Roussinov

A Dissertation Submitted To The Faculty Of The

DEPARTMENT OF MANAGEMENT INFORMATION SYSTEMS

In Partial Fulfillment Of The Requirements

For The Degree Of

DOCTOR OF PHILOSOPHY

WITH A MAJOR IN INFORMATION SYSTEMS

In The Graduate College

THE UNIVERSITY OF ARIZONA

1 9 9 9

Final Examining Committee Approval Form

STATEMENT BY AUTHOR

This dissertation has been submitted in partial fulfillment of requirements for an advanced degree at The University of Arizona and is deposited in the University Library to be made available to borrowers under rules of the Library. Brief quotations from this dissertation are allowable without special permission, provided that accurate acknowledgment of source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the head of the major department or the Dean of the Graduate College when in his or her judgment the proposed use of the material is in the interests of scholarship. In all other instances, however, permission must be obtained from the author.

SIGNED: _____

ACKNOWLEDGEMENTS

This research would not have been possible without funding from the National Science Foundation (NSF) and DARPA (??) and the generous support and encouragement of some key individuals and their institutions. At the University of Arizona, I am indebted to my committee members -- Dr. Jay Nunamaker, director of The Center for the Management of Information, Dr. Olivia Sheng, heading the Management Information Systems department, and to the entire team of faculty members for wonderful creative environment and valuable comments, in particular to Dr. S. Weisband, Dr. D. Meader, Dr. A. Datta.

My work is a part of a larger stream of research in Artificial Intelligence Laboratory under the direction of Dr. Hsinchun Chen who not only provided a source of funding for my program but also ignited my desire to become a researcher. He and Dr. B. Schatz from the University of Illinois at Urbana Champaign have been in charge of Digital Library project, part NSF/DARPA Digital Library Initiative which has funded most of my dissertation research.

All of my interactions with participants in this project were interesting and intellectually stimulating. I am especially indebted for code, expertise and idea sharing to M. Ramsey, D. Ng, M. McQuaid, and K. Tolle.

Dr. Chen not only provided research support and an excellent group of colleagues, but he also provided many publishing opportunities and strongly encouraged me in all of my publication efforts successful and unsuccessful. Furthermore, he introduced me to Mrs. Barbara Sears. This wonderful woman has edited many one of my publications and current dissertation we well.

I am extremely fortunate in having my dearest and devoted friend Lena who not only supported me with her love and care throughout the years but also endured editing many drafts of this work. There is no way I could possibly have started and accomplished this without her.

I am grateful to my former associates in software industry for giving me valuable experience and understanding my desire for an advanced degree, in particular R. Reed, CEO of Coextant Systems Co.

Many faculty members from the institutions that invited me for an interview while recruiting for an assistant professor position have also continued with their valuable comments, including those from University of Michigan, British Columbia, Kansas, Concord, Syracuse, Georgia State and Oklahoma State Universities.

DEDICATION

To someone forever dear to me, my farther who passed away in 1999 with an unrealized wish that came one month too late.

TABLE OF CONTENTS

TABLE OF CONTENTS - *CONTINUED*

TABLE OF CONTENTS - *CONTINUED*

TABLE OF CONTENTS - *CONTINUED*

TABLE OF CONTENTS - *CONTINUED*

TABLE OF CONTENTS - *CONTINUED*

TABLE OF CONTENTS - *CONTINUED*

LIST OF FIGURES

LIST OF FIGURES - *CONTINUED*

LIST OF FIGURES - *CONTINUED*

LIST OF TABLES

ABSTRACT

Successful knowledge management requires efficient tools to manage information in the form of text. However, our productivity in generating information has exceeded our ability to process it, and the dream of creating an information-rich society has become a nightmare of information overload.

Although researchers and developers believe that interactive information access systems based on clustering and summarization offer a potential remedy to that problem, there is as yet no empirical evidence showing superiority of those tools over traditional keyword search.

This dissertation attempted to determine whether *automated clustering can help to find relevant information* by suggesting an innovative implementation and verifying its potential ability to be of help. Our implementation is based on Kohonen's self-organizing maps and acts as a visualization layer between the user and a keyword-based search engine. We used the clustering properties of self-organizing maps to create a *summary* of search results. The user relies on this summary when deciding whether and how to provide additional feedback to the system to obtain more relevant documents.

We have resolved multiple issues related to the speed and quality of output associated with self-organizing maps and created a version (Adaptive Search) that allows interactive Internet searching.

We have performed user studies and a controlled experiment in order to test the proposed approach. In a laboratory experiment, subjects spent *less time* finding correct answers using Adaptive Search than using the search engine directly. In addition, the documents containing answers were *positioned consistently higher* in the rank-ordered lists suggested by Adaptive Search as opposed to the lists suggested by the search engine. The search engine that we used was AltaVista, known to be one of the most popular, comprehensive and flexible engines on the Web.

Our main conclusion is that indeed *information clustering helps information seekers if properly implemented*.

1. INTRODUCTION

1.1 Background

In ‘80s and ‘90s the cost of storing information electronically went down considerably, so available collections of text and multimedia documents proliferated. Internet and the World Wide Web have made such collections readily available to the public. Communication technologies have contributed millions of electronic messages and sites representing a variety of interests. Personal home pages and business web sites have in recent years been accruing at a rate of hundreds of thousands a day.

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

Along with the Internet, there are other large depositories of text documents such as company intranets. Many companies have realized that their current knowledge, as represented by their employees’ experience, data, and values is an important competitive asset (Davenport & Prusak, 1998; Nonaka, 1994). Organizations already have in electronic format their email messages, memos, customer support records, meeting transcripts, and internal reports, and the ability to search, categorize and visualize this knowledge has been identified as a vital component in organizational knowledge management (O’Leary, 1998). Although the bulk of organizational data is currently in text format, the ability to search it remains inadequate (Gordon, 1997).

In the realm of the traditional keyword approach to information searching, we need to match words in the searched text with the words that we enter to the system. For example, finding the answer to the question “How long does it take to get by train from Copenhagen to Oslo?” seems possible through composing a query “copenhagen AND oslo AND train” in a boolean syntax. Being entered to AltaVista (www.altavista.com), one of the most popular Internet search engines, this query results in about 900 matching web pages, only one of which contains the answer. Most users tolerate exploring only the first 20-30 pages in the ranked lists returned by search engines and give up.

There are some additional reasons for the keyword approach to fail to cope with the volumes of information available nowadays: very few people know query languages or search tools well, so they usually have difficulty in composing efficient queries and there is a mismatch between human beings’ common sense and boolean logic (Hearst, 1999).

1.2 Visual Tools for Interactive Information Access based on Clustering and Summarization

Information foraging theory (Pirolli et al., 1995) notes that there is a trade-off between the value of information and time spent finding it. People read through textual information much more slowly than computers, which can sift through megabytes of text in a moment. Exhaustive manual exploration of all potentially relevant documents is time prohibitive for many human users.

Unfortunately for users, however, computers do not have the human notion of common sense (Simon, 1991). They do not understand natural language variations and cannot autonomously find answers to even unambiguous questions posed by human beings. Besides, people are often vague and need several reiterations to describe their information needs to other people. When users approach an information access system they often have only a fuzzy understanding of how they can achieve their goals (Ingwersen, 1994). So it is not surprising that iterative clarifying is required while interacting with a computer system (Rao et al., 1995). This explains growing interest in interactive information access tools and paradigms.

Recent advances in computational speed, graphical user interfaces, computer graphics and displaying devices resulted in the emerging field of *information visualization* (Tufte, 1990), which attempts to provide visual depictions of very large information spaces. A visual representation communicates information more effectively than any other method. We know that a picture is worth 1000 words. For example, a photograph is more meaningful than a description of a person's face.

An example of visualization is a colorful diagram of stock market movements, in which an image of peaks and valleys is much more comprehensible than a table. Visualization also helps demonstrate certain physical phenomena of which photographs cannot be taken, such as in medicine (something too small or obscured) or astronomy (something too large in scale). It consequently seems very logical to apply visualization tools to aid

navigation in information spaces in order to facilitate access to information encoded in text documents.

Many new interactive visual tools based on summarization and clustering are being developed and marketed by leading companies that believe such tools to be helpful in navigating large volumes of data (Hearst, 1999). For example, the Scatter/Gather by Xerox (www.parc.xerox.com) clusters text documents according to similarities between them and automatically computes an overview of the documents in each cluster.

Similar approaches employed by commercial software systems (e.g., *Hyperbolic Tree* by Inxight Software (www.inxight.com) or *SemioMap* by Semio Corp., (www.semio.com) allow users to navigate within automatically created categories of concepts (maps). Pacific Northwest National Labs (www.pnl.gov) has developed visualization tools based on two- and three-dimensional representations of document collections. The Illinois Digital Library Initiative project (www.forseti.grainger.uiuc.edu; Schatz & Chen et al., 1996) has adopted self-organizing maps (a clustering and dimension reduction technique) for visual displaying and categorizing millions of text documents.

1.3 The Scope of the Dissertation

So far, no experimental evidence has been tendered to show that an approach with some combination of the above characteristics leads to faster or more effective searching of a collection of documents (Hearst, 1999). Neither has there been any empirical study

evaluating the applicability of tools utilizing clustering for searching the World Wide Web or intranets.

This dissertation study answers the question *whether automated clustering of text documents can help interactive information seeking*. We have been particularly interested in strong effects, those that are potentially able to reduce Information Overload and be effectively used in Knowledge Management technologies.

We understand that there are many other potential approaches to reducing information overload if properly implemented in modern systems. Many of them have been suggested based on user observations and sometimes just common sense, including such convenient features as 1) *retention of context between searches* 2) increasing *interface usability* 3) *better user training, documentation and tutoring wizards*. However, we have chosen our research question due to its being challenging (there is not empirical evidence of success so far) and its promising to yield new information-seeking paradigms.

To address our research question, we have decided to follow the *system building methodology* (Nunamaker et al., 1991). We started by selecting and designing appropriate algorithms, proceeded to system building and finally empirically evaluated several proof-of-concept prototypes.

Since answering our research question negatively would require exhaustive exploration of all possible uses of clustering and thus would be cost prohibitive, we have sought to find a positive answer by discovering a use of clustering that helps in interactive information seeking. To address our research question we came up with a novel use of

clustering which was accompanied by user feedback and rank ordering of documents by relevance. Through this dissertation study, we have found that our suggested implementation (called *Adaptive Search*) significantly increases users' ability to find relevant documents compared with the traditional query based approach.

Adaptive Search uses Kohonen self-organizing maps (Kohonen, 1995) and acts as a layer between the user and a commercial keyword-based search engine. A Kohonen self-organizing map (SOM) is a relatively new and promising technique, patterned after the associative properties of human brain. It is an unsupervised neural network used for many clustering and dimension reduction applications. Developed in 80s, SOM has since been successfully used for dimension reduction and clustering in many applications such as pattern recognition and natural language processing.

Since SOM is the only clustering/summarization technique that we have used to address our central research question, the more accurate reformulation of that question is: *whether a self-organizing map can help interactive information seeking* more effectively than using an Internet search engine directly.

We used SOM to visualize query results by taking advantage of its clustering and summarization abilities. While using Adaptive Search, the interaction among the user, the system and the search engine consists of the following steps:

Step 1. The user submits a query expressed in a boolean language or as natural text.

Step 2. The search engine returns a list in ranked order of documents matching the query.

Step 3. Adaptive Search builds a self-organizing map for the 200 top documents in the ordered list.

Step 4. Based on the map, the user may change the query and return to step 2 or request an ordered list of documents resulting from the user feedback.

Step 5. The user browses the documents in the ordered list to find those of interest.

More details follow in chapters 4 and 5.

From this implementation, several research questions surfaced in addition to the central one. Clustering and neural network algorithms have been notoriously slow. Users rarely have enough patience to wait while the system is doing something intelligent. So, there is another question to answer: "*Is fast and scalable implementation based on a self-organizing map algorithm possible?*"

Generally, SOM is a clustering tool, but its ability to cluster text documents (not just terms) has never been empirically evaluated in text analysis applications. So another question to answer is: *Are SOM clustering abilities good enough to create concepts making sense to users?*

1.4 Dissertation Structure and Writing Style

This dissertation research consists of four studies, all related to applications of self-organizing maps to text search and visualization tasks. The first study has resolved many

issues related to the speed and scalability of a self-organizing algorithm, which is crucial for interactive real-time applications.

The second explored clustering abilities of SOM in the domain of textual documents. Our third study has resolved several problems with output quality of self-organizing maps by suggesting a customizable implementation of SOM. For this study, we also built an interactive prototype search system and observed its users' behavior.

Finally, building upon the results of the first three studies, we have developed and empirically evaluated our Adaptive Search prototype system and arrived at a positive result: Our implementation based on the self-organizing map as a clustering and summarization tool indeed substantially helped information seekers.

The next chapter presents a literature review covering the entire scope of this dissertation and our research formulation. Four chapters, each corresponding to a study follow, containing additional literature review specific to each chapter.

Although the author is accountable for 95% of the work reported here, he acknowledges contributions by the entire team of the Artificial Intelligence Laboratory, led by McClelland and Andersen Consulting Professor Hsinchun Chen, by using the pronoun "we" instead of "I" throughout the dissertation.

2. LITERATURE REVIEW AND RESEARCH FORMULATION

This chapter provides an overview of reported prior work related to the entire scope of this dissertation. Each chapter describing a separate dissertation study has additional references to the literature pertinent to that particular study. This chapter starts with a description of the challenges to developing knowledge management technologies, in particular the problem of information overload. Then we describe prior attempts to alleviate that problem through automated document clustering and summarization. An overview of Kohonen's self-organizing map technique follows. At the end of the chapter, we summarize problems that still remain and present our research formulation, which includes our research questions and methodology.

2.1 Knowledge Management and Information Overload

Knowledge management researchers define "knowledge" as *experience, values, and data* that organizations possess (Davenport & Prusak, 1998; Nonaka, 1994). They have noticed that in today's overheated job market employee turnover is high, so knowledge becomes a corporation's biggest competitive advantage. People come and go but the knowledge stays in the organization.

O'Leary (1998) has noticed that organizational knowledge management is becoming so important that 40% of fortune 1000 companies have CKO-s – Chief Knowledge Officers. O'Leary has also noted that tools to categorize, search and visualize knowledge are extremely important for successful knowledge management.

As firms rely increasingly on intranets (Wagner & Engelmann, 1997), enterprise information resources proliferate in forms already familiar on the WWW (Bannan, 1997). In particular, firms following an intranet personalization strategy (Hansen et al., 1999) tend toward less codified databases and less structured information resources and stronger resemblance to the WWW as a whole. Opportunities for knowledge management in the intranet context also have been proposed (Ba et al., 1997; Davenport & Prusak, 1998; Nonaka, 1994).

Gordon (1997) has found that 80-90% of organizational data is in the form of unstructured text. The only technologies that have been traditionally applied to searching, categorizing and visualizing unstructured textual data are Information Retrieval (IR) technologies (Salton, 1983). IR technologies rely on exact or partial word match between the keywords associated with text documents and the query words entered by users.

Current information access systems and models assume an interaction cycle consisting of query specification, receipt and examination of retrieval results, and then either stopping or reformulating the query and repeating the process until a satisfactory result set is found (Hearst, 1999; Salton, 1989; Shneiderman et al., 1998). In more detail, the standard process can be described according to the following sequence of steps:

1. Start with an information need

2. Select a system and collections to search on

3. Formulate a query

4. Send the query to the system

5. Receive the results in the form of information items

6. Scan, evaluate, and interpret the results

7. Either stop or

8. Reformulate the query and go to step 5.

The ability to apply IR technologies to organizational knowledge repositories remains inadequate (Gordon, 1997) mostly due to the associated problem of Information Overload. Blair and Maron (1985) has defined this condition as "Output Overload," the problem of overwhelming users with too many potentially relevant documents (on step 6) as a result of low accuracy of retrieval results. This problem occurs not only in organizational settings but virtually in all cases when large repositories are involved, including the case of World Wide Web (Hearst, 1997).

Researchers have held many problems accountable for low accuracy of retrieval results. Furnas et al. (1987) has identified the so called "vocabulary problem" by demonstrating that people use the same words to describe same information in only 20% of cases. Besides, there is inherent imprecision in human language: Same meanings can be expressed by different words (a problem of *synonymy*) and words may have several meanings (a problem of *polysemy*) (Bates, 1986; Bartell et al., 1995).

Another problem is that very few users know how to use each technologies well (Hearst, 1997). Chen and Dhar (1990) have noticed that for successful information finding the

user must have a working knowledge of the system where the information is stored, in particular how to navigate through that system and what terminology to use. Very few users have this type of knowledge.

Most users have great difficulty specifying queries in Boolean format and often misjudge what the results will be (Hearst, 1999; Young & Shneiderman, 1993). Boolean queries may be problematic for several reasons. Most people find the basic syntax counter-intuitive. Many users confuse everyday semantics associated with Boolean operators. For example, to inexperienced users, using AND implies the widening of the scope of the query: ``stocks AND bonds" may be interpreted as a request for documents about “stocks” and documents about “bonds,” rather than documents that talk about both “stocks” and “bonds.”

2.2 Interactive Search Tools based on Clustering and Summarization

Everitt (1974) defined a cluster as "a set of entities which are alike, and entities from different clusters are not alike." An example of an early study on clustering in Information Science is work by Jardine and van Rijsbergen (1971). The idea behind clustering was that if certain documents match a user query, the documents in the same cluster also are likely to be relevant. This has been traditionally known in information science as the Clustering Hypothesis. However, experiments in which an entire collection of documents was organized into a static cluster hierarchy did not demonstrate any improvements in the relevance of the retrieved documents (Croft, 1995; Voorhees, 1985; Willett, 1988; Hearst, 1999a).

A good overview of use of clustering applications in automated (non-interactive) information retrieval has been done by Rasmussen (1992). She identified the following major problems with applying various clustering techniques in the text analysis domain:

1. Difficulty of assessing the validity of results obtained.

2. Selecting appropriate attributes to represent text documents.

3. Selecting an appropriate clustering method.

4. High cost in terms of computational resources.

Shneiderman (Shneiderman, 1996) recommended an interaction model in which the user begins with an automatically created overview of the information, then zooms to find areas of potential interest.

Many interactive information access systems rely on document clustering. The idea behind them is to agglomerate similar documents into clusters and present a high-level summary of each cluster. This way, the user does not need to go through similar documents or through entire documents in order to become familiar with the collection. This greatly reduces redundancy and cognitive demand. Examples of such visualization systems are Scatter/Gather (Cutting et al., 1992), WebBook (Card et al., 1996), and SenseMaker (Wang, 1997). Hearst (1999) gives a good overview of such systems and the ideas behind them. She also has noted that so far the empirical evidence that any of those tools facilitates information access has yet to be presented.

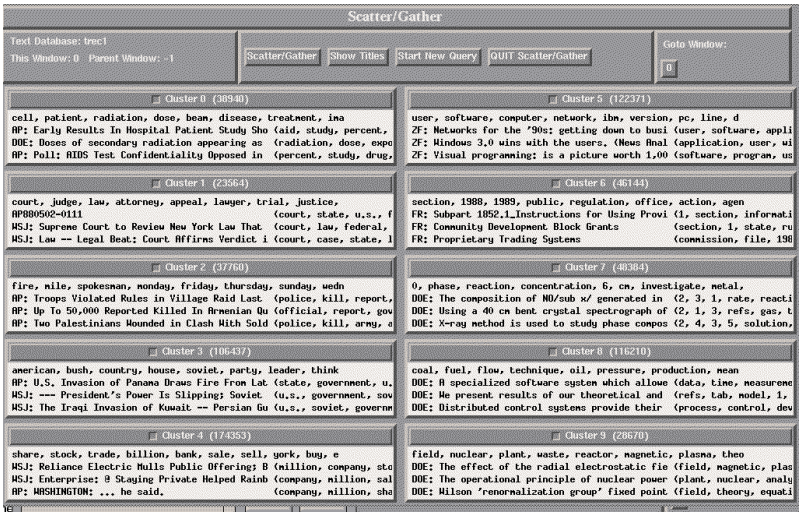


Figure 2.1. The Scatter/Gather system.

The Scatter/Gather system, shown on Figure 2.1, clusters documents into groups, and presents descriptive textual summaries to the user (Cutting et al., 1992). The summaries consist of topical terms that characterize each cluster generally, and a set of typical titles that hint at the contents of the cluster. The user may select a subset of clusters that seem to be of interest and re-cluster their contents, thus examining the contents of each sub-collection at a finer granularity.

By running experiments with subjects, Pirolli et al. (1996) have found that Scatter/Gather indeed communicates a high-level picture of a collection of documents. They asked subjects to perform specified search tasks and then to draw a hierarchy of concepts into which they would identify items in the collection. The subjects who used Scatter/Gather were in much closer agreement about their understanding of topics and also provided richer descriptions of the collection than the subjects who used keyword searches. However, the subjects required more time to perform the search tasks with Scatter/Gather

and the documents found were judged less relevant. The study concluded that search based on exploration of a hierarchy of clusters was not superior to the one that was keyword based but suggested that a combination of clustering and keyword search might be superior to both.

Hearst and Pedersen (1996) integrated Scatter/Gather with conventional search technology by clustering the results of a query. They found that relevant documents tended to fall mainly into one or two out of five clusters, and that the precision and recall were higher within the best cluster than within the retrieval results as a whole. On average, 60-100% of the relevant documents appeared in the “best cluster” and approximately 80-100% in the two best ones. The implication of this finding is that entire set of documents can be filtered out by the user immediately and be ignored as irrelevant. A user might save significant time by looking only at the contents of the seemingly most promising clusters.

Hearst (1999) has noted that “the unsupervised nature of clustering can result in a display of topics at varying levels of description.” For instance, clustering a collection of documents about the stock market might result in 5 clusters: documents about stock quotes, mutual funds, market indices, electronic commerce, government policies and legal issues. The last two are apparently more general than the first three, since they discuss issues not limited to the stock market.

2.3 Kohonen's Self-Organizing Maps

Our study used Kohonen's self-organizing map (SOM) (Kohonen, 1995) for clustering, summarization and eliciting user feedback. SOM is a kind of unsupervised neural network. Developed in the 1980s, SOM has since been successfully used for dimension reduction and clustering as well as in many applications such as pattern recognition and natural language processing. An advantage of SOM over other clustering algorithms is its ability to visualize high dimensional data using a two-dimensional grid while preserving similarity between data points as much as possible. It is a similar technique to Multidimensional Scaling (MDS) (Jain & Dubes, 1988). In SOM, each input node corresponds to an input dimension. Each output node corresponds to a node in a two-dimensional grid. The network is fully connected in the sense that every mapping node is connected to every input node with some connection weight. During the training phase, the inputs are presented several times in order to train the connection weights in such a way that distribution of output nodes represents distribution of input points. The network trains fully automatically, without any human intervention. The topology of the Kohonen SOM network is shown in Figure 2.2. More details on the SOM algorithm follow in the "Scalable SOM Study" chapter.

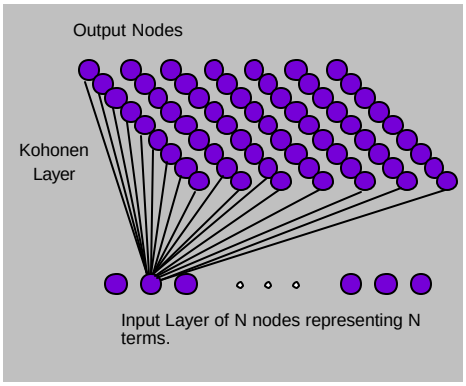


Figure 2.2. Kohonen SOM network topology.

Many engineering and scientific applications which involve numeric data (e.g., image recognition, signal processing) have successfully adopted the SOM approach to parallel clustering (Kohonen, 1995). Ritter and Kohonen (Ritter & Kohonen, 1989) applied the Kohonen SOM to textual analysis in an attempt to detect the logical similarity between words from the statistics of their contexts. The Kohonen group has created and maintained a WEBSOM server that demonstrates its ability to categorize several thousand Internet newsgroup items (Honkela et al., 1996).

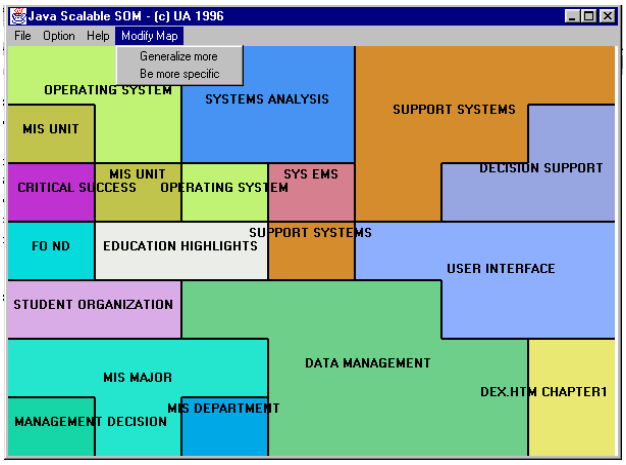


Figure 2.3. Visual representation of clusters of documents by a Kohonen's self-organizing map.

Chen et al. (1996) used Kohonen's feature map algorithm to create graphical maps that characterize the content of a document collection (see Figure 2.3). The regions of the 2D map, varying in size and shape, correspond to clusters of documents within the collection. Regions are characterized by labels composed of single words or phrases frequently occurring within the documents assigned to each particular region. Due to similarity preservation, adjacent regions are semantically related. The regions can be nested (hierarchical) if they contain enough documents.

The Illinois Digital Library Initiative project (Schatz & Chen, 1996) has adopted self-organizing maps for visual display and categorizing of millions of text documents. Chen et al. (1998) compared browsing self-organizing document maps to browsing Yahoo Internet directory. Subjects were asked to find an ``interesting" web page within the entertainment category of Yahoo and also within SOM. The study concluded that the

SOM interface was more appropriate for casual browsing than for search and also that SOM gave users a ”gist” of the kinds of information within the document collection. They noticed problems related to the confusing different level of topic granularity.

Orwig et al. (1997) described research in the application of an SOM algorithm to the problem of classification of electronic brainstorming output and evaluation of the results by comparing SOM output with Hopfield neural network (Chen et al., 1994) and human expert output. They found that the SOM performed as well as a human expert in identifying key concepts in the meeting output and out-performed the Hopfield neural network algorithm.

One of the major drawbacks of neural network computation, including the SOM algorithm, has been its computational complexity. Training instances are often presented multiple times and network performance is achieved only after gradual modification of network connection/link weights. Although the categorization robustness and graphical friendliness of the SOM-family of algorithms have been shown in several studies, the computational complexity of such algorithms has caused severe implementation problems, especially for mid-to-large-scale applications. A mid-scale application such as the Internet homepage categorization project reported in (Orwig et al., 1997) and involving clustering 10,000 homepages required about 10 hours of processing on a DEC Alpha 3000/600 workstation (200 MHz, 128 MBs RAM). The computational complexity of the SOM algorithm has rendered it infeasible for large-scale (1-10 GBs, millions of documents, e.g., the entire searchable Internet WWW homepages) or interactive real-time

applications. In order to improve the scalability of the SOM approach to textual classification, a more efficient algorithm is needed.

2.4 Remaining Problems

Below, we summarize the problems indicated in prior research that we address in this dissertation study:

1. **The absence of empirical evidence.** In spite of the existence of algorithms able to cluster text document according to their similarity in such a way that the resulting clusters match human expectations, the interactive information access systems based on clustering *have yet to be shown to be effective.* Hearst (1999) wrote:

"We do not have evidence at this time about whether users are better off seeing clusters or groups of categories, whether users can recover well from confusing clustering, whether the arbitrary element of clustering outweighs the potential usefulness of its ability to find strong themes."

2. **Quality.** The output of the self-organizing map algorithm does not always match user's expectations (Chen et al., 1996). As a result, it is not known whether SOM can aid interactive information access.

3. **Speed.** Automatic clustering, and Kohonen's self-organizing maps in particular, are known to be very time consuming (Kohonen, 1995). As a result, it is not known whether they can be implemented efficiently enough to be used for real-time interactive information access.

4. **Clustering properties not validated.** Although clustering and dimension reductions of Kohonen's self-organizing maps have been extensively studied in other domains, and its ability to cluster terms found in text documents (concepts) has been empirically studied (Orwig et al., 1997), its ability to cluster documents (not terms) has not been yet directly evaluated.

2.5 Research Formulation

This section starts with a review of our research questions and methodologies. Then, we establish a premise for considering our *Adaptive Search* more effective than the prior implementations of interactive search systems based on clustering and more effective than the traditional keyword search. Chapter 6 gives more technical details about this approach. The current section also explains how all four dissertation studies relate to each other.

2.5.1 Research Questions and Methodologies

This dissertation study answers the question *whether automated clustering of text documents can help interactive information seeking*. To address this research question, we have decided to follow the system development methodology (Nunamaker et al., 1991), in which a problem is identified and an automatic or system-based solution is designed, programmed and implemented, typically using one of the systems analysis and design paradigms. The resulting system solution is then tested first in a laboratory setting, and then later "out in the field" or in a real-world setting. We started by selecting and

designing appropriate algorithms, proceeded to system building and finally empirically evaluated several proof-of-concept prototypes.

Since answering our research question negatively would require exhaustive exploration of all possible uses of clustering and thus would be cost prohibitive, we have sought to find a positive answer by discovering a use of clustering that helps in interactive information seeking.

To address our research question we performed a sequence of four studies and came up with a novel use of clustering (called Adaptive Search), which acts as a *summary* and incorporates *user feedback* and *rank ordering* of documents by relevance.

Adaptive Search uses Kohonen self-organizing maps (Kohonen, 1995) and acts as a layer between the user and a commercial keyword-based search engine. More details about SOM were presented in the preceding section of this chapter. Since SOM is the only technique that we used to address our central research question, a more accurate reformulation of it would be: *whether a self-organizing map can help interactive information seeking*.

2.5.2 Adaptive Search Approach and Prior Research

Although our approach goes deeper than simply changing the user interface (we are suggesting new features), we have found GUI research quite inspiring. Shneiderman (1997) lists principles for good user interfaces design, including providing *informative*

feedback, *reversal of actions*, *reducing working memory load*, and providing alternatives for *novice and expert users*.

To achieve this, we have followed a previously suggested idea of visualizing current query results (Hearst & Pedersen, 1996; Plaisant et al., 1997), which we accomplish by automated clustering and summarization. We argue that clustering query results has an advantage over clustering an entire collection since it is specific to the task at hand. In a way, it *adapts* to a particular user (novice or expert) and a particular search task.

Judging from the experience of other researchers mentioned in the preceding sections, we can hypothesize that document clustering may reduce cognitive effort that otherwise would be necessary to inspect each stand-alone document. Based on the automatically computed overview of each cluster, the user may at once discard multiple documents as irrelevant.

Studies have shown that users may easily get disoriented while navigating hierarchies, both manually created such as Yahoo as well as automatically generated such as self-organizing maps (Chen et al., 1996). Navigating requires remembering which path is being currently pursued. It may be time consuming and frustrating if the desired information is deep in the hierarchy or not available at all. A wrong path requires backing up and trying again. On the other hand, presenting documents in rank order (Salton, 1989) has become standard for modern search systems including Internet spiders. This is why we decided to get away from hierarchical exploration and return to rank ordered lists.

We argue that since clusters effectively serve as a summary of a set of text documents, the user may be able to modify the query or provide some other feedback to the system by looking at those clusters without actually going through time consuming exploration of documents. Adaptive Search allows three major types of feedback: 1) selecting what is relevant 2) rejecting what is not relevant and 3) specifying what is missing in a summary.

The application of Kohonen's self-organizing map (Kohonen, 1995) to visualizing text data possesses the necessary properties: 1) the algorithm clusters documents into regions, 2) it serves as a summary of them, and 3) it identifies key concepts in those documents. We decided to use the key concepts as means of providing feedback to the system. Suggesting terms to the user to expand the query has been found helpful to information seeking process (Salton, 1989; Schatz & Chen, 1996).

Since our approach to using clustering for interactive information access was different from those reported in prior studies and also incorporated many suggestions from them, we hoped that it might result in better results than prior studies had reported. We hoped that our users would spend less time seeking the relevant information and that a higher proportion of search sessions would be successful.

Adaptive Search does not require familiarity with a query language. It does not even inform the user what queries it creates in order to obtain the final rank-ordered list of documents. This way, our approach is different from query formulation interfaces (Pedersen, 1993; Landauer et al., 1993). We think of our approach as a first step toward a more general *information navigation* paradigm, in which the user only directs the system

toward the documents that satisfy the information need, and does not need to think in terms of boolean queries.

2.5.3 Adaptive Search and Dissertation Studies

Figure 2.4 displays a diagram showing necessary properties and features of an interactive search system the way we envisioned it and how our dissertation studies have addressed issues related to them. As we indicated above, the system is based on three major

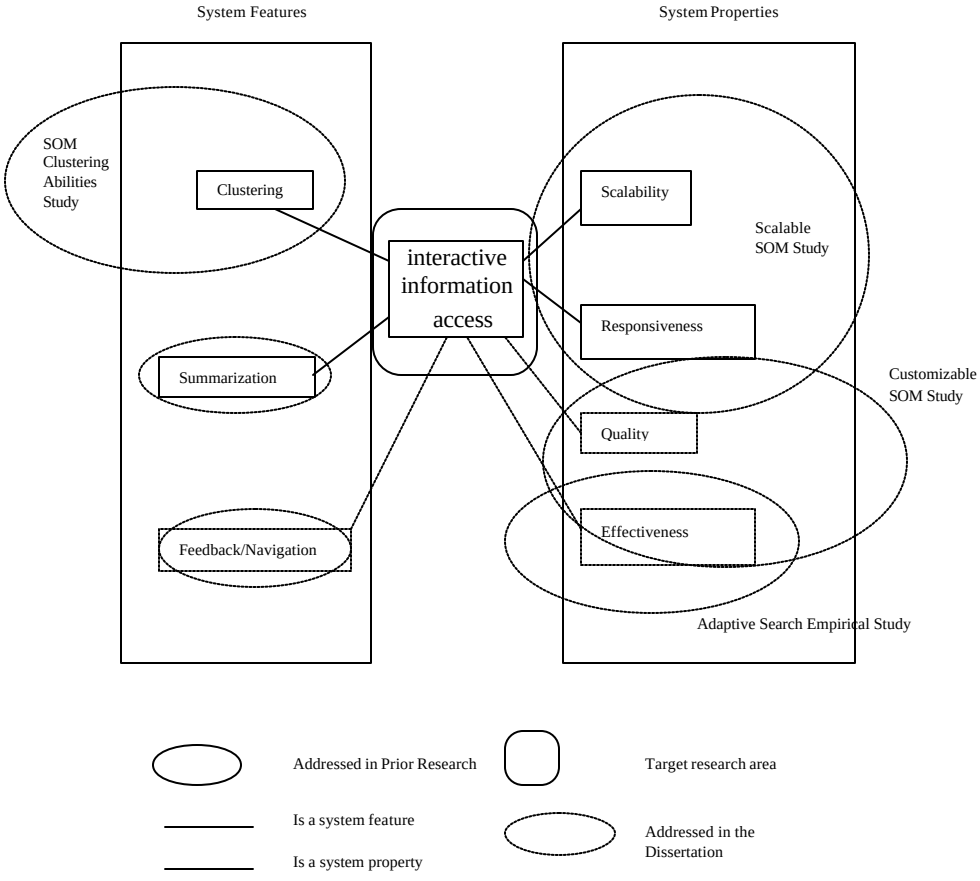


Figure 2.4. The properties and features of our hypothetical interactive search system.

features: document *clustering*, cluster *summarizing*, and ability to accept user *feedback*. Our "SOM Clustering Abilities" study has tested whether SOM can be used as document clustering tool. Since interactive real-time systems must be fast and responsive, we tried to address most of the issues related to SOM speed in our "Scalable SOM" study. We have explored potential solutions to the problems reported with the quality of output of most automated text clustering tools in our "Customizable SOM" study, which also has touched upon some evaluation issues through a user study. Finally, we empirically addressed our central research question in an empirical "SOM and Internet Search" study.

3. SCALABLE SOM

3.1 Objectives

Interactive real-time system must be fast and responsive, since the users rarely have time and patience to wait long for a system's response. Both clustering and neural network algorithms have been known to be notoriously slow (Rasmussen, 1992; Pirolli et al., 1996; Kohonen, 1995). This chapter presents a study that has explored the possibility of creating a fast and scalable algorithm, adequate for quick interactive clustering of text documents.

We have proposed a much faster and scalable implementation of Kohonen's self-organizing map (SOM) algorithm for the domain of text analysis. Our proposed data structure and algorithm took advantage of the *sparsity* of coordinates in the document input vectors and reduced the SOM computational complexity by several order of magnitude, while generating the same output as the original algorithm.

Algorithmic intuition and the mathematical foundation of our research are presented in detail. We also describe three benchmarking experiments to examine the algorithm's performance at various scales: classification of electronic meeting comments, Internet homepages, and the Compendex collection.

3.2 Research Questions and Methodology

By this study, we answered the question: *Is fast and scalable implementation of Kohonen's self organizing maps possible?* Being fast implies speed enough faster than that of the original Kohonen's algorithm (Kohonen, 1995) to be used in interactive clustering of query results. The scalability property implies that the computational requirements do not increase more than proportionally to the size of the task so that the technology can be extended to larger scale applications.

We have followed the system building methodology (Nunamaker et al., 1991). We started from designing an algorithm, proceeded to prototype implementation and run several benchmarking experiments to evaluate computational requirements such as processing time and computer memory. We have measured processing time as a function of the input vector size, which is the number of terms (words or phrases) used to represent the documents in the collection.

3.3 Background and Issues

As we wrote in our introduction, one of the major drawbacks of neural network computation, including the SOM algorithm, has been its computational complexity. Training instances are usually presented multiple times and network performance is achieved only after gradual modification of network connection/link weights. The prior experience in adopting SOM in several mid-size information visualization and categorization projects (10-100 MBs, several to hundreds of thousands of abstract-size

documents) confirmed this general observation (Chen et al., 1996). The computational complexity of the SOM algorithm has rendered it infeasible for large-scale applications (1-10 GBs, millions of documents, e.g., the entire searchable Internet homepages).

Orwig et al. (1997) applied SOM to categorizing the output from electronic brainstorming meetings. The participants had to take a 20-30 minute coffee break during which the slow SOM algorithm was crunching numbers. The Illinois Digital Library Initiative project that has adopted self-organizing maps for visual displaying and categorizing millions of text documents had to use hours of supercomputer time at the National Center for Supercomputing Applications (Schatz & Chen, 1996).

Kohonen's group at the Helsinki University of Technology used special parallel hardware to keep computational time at a manageable level (Kohonen, 1995). They recently started working on algorithmic optimizations as well by exploring the idea of a preliminary dimension reduction (Kaski et al., 1998) through a novel technique called *random mapping*.

3.4 Testbed

We have run three benchmarking experiments to examine the algorithm's performance at various scales: classification of electronic meeting comments (small), Internet homepages (intermediate), and the Compendex collection (large). This section describes the collections used. Section 3.6 reports the results.

3.4.1 Electronic Brainstorming Comments

Orwig et al. (1997) described research in the application of an SOM algorithm to the problem of classification of electronic brainstorming output and evaluation of the results. Electronic brainstorming is one of the most productive tools in the electronic meeting system called GroupSystems (Nunamaker et al., 1991a). A major step in group problem-solving involves the classification of electronic brainstorming output into a manageable list of concepts, topics, or issues that can be further evaluated by the group. Information overload and the cognitive demand of processing a large quantity of textual data make this step problematic.

EBS comments exhibit some unique characteristics and often contain typos, abbreviations, and incomplete sentences. Typically, a one-hour session with a dozen or so participants generates several hundred comments. The sample electronic brainstorming data set for our SSOM bench-marking experiment consisted of 202 small paragraph-size comments. Figure 3.1 displays several sample comments. Due to its small size this collection models small scale tasks.

Currently collaborative rooms are expensive to set up and maintain. We need to be able to do portable environments in settings where we would have normal meetings. This would entail the use of wireless lan technology that work beyond the line of site in the current technology

Cellular technology hold great promise for having remote Collaborative meetings but the reliability and band with need to be Improved.

The MOST important is to somehow make sure that the technology does not overtake the human part of human interaction.

User awareness of the capabilities and benefits of using this Technology----"Getting the word out".

Repository technology will have to move toward an acceptable Standard that will promote optimal sharing between various types of environments. Don't make our own standards.

Understanding how our technology encapsulates specific cultural Expectations -- and learning which technologies are appropriate in which cultures.

How to measure the validity and reliability of groupware tools

We need to build "techniques" or "processes" on top of the raw Tools we are currently developing for electronic group settings.

Figure 3.1. Sample EBS comments

3.4.2 Internet Entertainment Homepages

The problems of information overload and vocabulary differences have become more pressing with the emergence of the increasingly popular Internet services. As we

mentioned in our literature review, current Internet search overwhelms surfers with an abundance of irrelevant information.

The main information retrieval mechanisms provided by the prevailing Internet WWW software are based on either keyword search, such as Lycos (www.lycos.com) and AltaVista (www.altavista.com) or directory browsing such as Yahoo! (www.yahoo.com). Already mentioned in our literature review, research by Chen et al. (1996) aimed at providing an alternative concept-based categorization and search capability for WWW servers based on the SOM algorithm. They reported results of testing multi-layered SOM (MSOM) clustering algorithm to classify Internet homepages according to their content. The category hierarchies created could serve to partition the vast Internet services into subject-specific categories and databases and improve Internet keyword searching and/or browsing.

In this study we used the same collection of about 10,000 entertainment-related homepages (extracted by a spider running on the entertainment portion of the Yahoo! directory). We considered this size intermediate.

3.4.3 Compendex Abstracts

We used the documents from the very well known COMPENDEX collection, which consists of abstracts from such computing-related fields as electrical engineering, computer science, information systems etc. The collection had 247,721 documents and the size of the collection after we applied *indexing* (page 51) was 360 megabytes. After

indexing, we extracted about 160,000 unique terms, not counting words in a stop-word list or words that appeared fewer than three times. Of those, we selected 10,000 frequently occurring terms as the most representative. At the time of that study, this was to our knowledge the largest known collection processed by SOM algorithm.

3.5 Algorithms and Implementations

3.5.1 SOM for Text Analysis

In our studies reported in this dissertation, we proceeded through the standard sequence of automated text analysis (Salton, 1983), using the code developed for the study by Orwig et al. (1997). We overview below the sequence of steps and provide more details in subsequent sections. We followed the most commonly accepted Vector Space model (Salton, 1983) by performing the following steps:

Step 1. Automatic indexing.

Step 2. Creating document representation in the vector space.

Step 3. Processing document vectors, which in our case meant running the self-organizing algorithm. In general, it could be another kind of clustering or categorization algorithm.

3.5.1.1 Automatic Indexing

The purpose of automatic indexing is to identify the content of each textual document automatically by sets of associated features (Salton, 1983). Features are words and

phrases. Automatic indexing first extracts a set of all words and possible phrases that enter the document. It follows simple heuristic rules such as looking for white space characters (e.g. , “new line,” “tab,” or “space”) as word boundaries.

Then it removes words from a ``stop-word” list to eliminate non-semantic bearing words such as *the*, *a*, *on*, *in*. Our automatic indexing program also creates phrases from adjacent words. Single words or phrases used for document representation are called *terms.* In this research, we used the automatic indexing described in (Orwig et al., 1997).

For example, indexing the sentence “What is a linear thread meeting, why don't we want it?” may result in identifying the terms “linear,” “thread,” “meeting,” and “linear thread.”

3.5.1.2 Document Representation

A vector space model (Salton, 1983) represents each document as a point in a highly dimensional Euclidean space based on the words and phrases (called *terms*) that those documents have. For computational efficiency and accuracy of representation, we preserved only the top specified number of most frequently included terms in the collection.

Each coordinate in a vector space corresponds to a term. In our case, if a term did not enter the document, the corresponding coordinates set to 0. If a term entered the document, we set the coordinate to 1. Prior research has shown this scheme to be

adequate for electronic meeting messages (Orwig et al., 1997) and Internet home pages (Chen et al., 1996).

If in the example from the preceding subsection only the terms “linear.” “thread,” “facilitator,“ and “meeting” from the entire collection were used in the order listed to represent the documents, the representation of the sentence in the example would be (1 1 0 1).

3.5.1.3 Kohonen’s SOM Algorithm

The proposed SSOM algorithm is based on the conventional SOM algorithm developed by Kohonen (Kohonen, 1989). A sketch of a revised SOM algorithm for textual classification (Orwig et al., 1997; Chen et al., 1996) is summarized below:

1. **Initialize input nodes, output nodes, and connection weights:** Use the top (most frequently occurring) N terms as the input vector and create a two-dimensional map (grid) of M output nodes (say a 20-by-10 map of 200 nodes). Initialize weights w_{ij} from N input nodes to M output nodes to small random values.

2. **Present each document in order:** Describe each document as an input vector of N coordinates. Set a coordinate to 1 if the document has the corresponding term and to 0 if there is no such term. Each document is presented to the system several times.

3. **Compute distance to all nodes:** Compute Euclidean distance d_j between the input vector and each output node j:

$$d_j = \sum_{i=0}^{N-1} (x_i(t) - w_{ij}(t))^2 \quad (1)$$

where $x_i(t)$ can be 1 or 0 depending on the presence of *i*-th term in the document presented at time *t*. Here, w_{ij} is the vector representing position of the map node *j* in the document vector space. From a neural net perspective, it can also be interpreted as the weight from input node *i* to the output node *j*

4. **Select winning node j^* and update weights to node j^* and its neighbors:** Select winning node j^*, which produces minimum d_j. Update weights to nodes j^* and its neighbors to reduce the distances between them and the input vector $x_i(t)$:

$$w_{ij}(t+1) = w_{ij}(t) + h(t)(x_i(t+1) - w_{ij}(t)) \quad (2)$$

After such updates, nodes in the neighborhood of j^* become more similar to the input vector $x_i(t)$. Here, $h(t)$ is an error-adjusting coefficient ($0 < h(t) < 1$) that decreases over time. (See (Kohonen, 1989) for the algorithmic details of neighborhood selection and adjustment.)

5. **Label regions in map:** After the network is trained through repeated presentations of all documents (each document is presented at least five times), assign a term to each output node by choosing the one corresponding to the largest weight (*winning term*). Neighboring nodes that contain the same winning terms are merged to form a concept/topic region (group). Similarly, submit each document as input to the trained network again and assign it to a particular concept in the map. The resulting map thus represents regions of important terms/concepts with the documents assigned to them.

Concept regions that are similar (conceptually) appear in the same neighborhood. Similar documents are assigned into same or similar concepts.

3.5.2 Analysis of the Original SOM Algorithm

It is apparent that steps 2-4 from the preceding subsection are repeated many times for each document and thus account for most of the processing time required. Steps 3 (compute distance to all nodes) and 4 (update weights) require iterations through all coordinates in the input vector. The processing time T for the conventional SOM is proportional to the number of document presentation cycles (Step 2) and the vector size:

$$T = O(NC),$$

where N is the input vector size and C is the number of document presentation cycles.

For textual categorization, input vector size can be as large as the total number of unique terms in the entire collection. In previous experiments (Orwig et al., 1997; Chen et al., 1996), it was found that the number of unique terms for mid-scale collections (10-50 MBs) after applying thresholds (for example, a term appearing more than two times) can be as high as tens or hundreds of thousands. By observing the document collections that we have analyzed and that have been mentioned in the literature (Grefenstette, 1994), we concluded that the number of unique terms in a collection is typically proportional to the size of a collection. Even after selecting a smaller number of the most representative terms, as was done in (Orwig et al., 1997; Chen et al., 1996), in order to make the

representation meaningful, this number (vector size) has to grow in concert almost linearly with the collection size.

Representing the size of a collection as S, we can define N in terms of S as $N = O(S)$. Similarly, because each document is presented multiple times, C can be represented by S as $C = O(S)$. Thus, the total processing time T could be estimated as:

$$T = O(NC) = O(S^2)$$

Given the fact that the size of text collections targeted by automatic processing grows exponentially over time, we fear that even continuously improving computer hardware (e.g., parallel supercomputers) will not be able to build SOMs for text document collections using the traditional approach. The conventional SOM's time complexity of ``square of the size of collection'' is deemed not scalable. We believe that even parallel processing on the prevailing supercomputers (Chen et al., 1996a) will not resolve this algorithmic problem.

3.5.3 Intuition Behind our Modification

Based on our analysis of the characteristics of the representation, we noticed that sparsity of the input vector stood out as a prime candidate for SOM optimization. It is evident that for large-scale textual collections a vast majority of coordinates in the input vectors were *zeros* (non-existence of a given term) and only very few were *ones* (existence of a given term). Due to the large number of unique terms in mid-to-large-scale collections (and thus large input vector size), almost all documents could be represented by a sparse

vector of a few *1*-s, plus mostly *0*-s. Sparse vectors or matrices are often candidates for optimization in software engineering and operations research. We also noted that such an optimization technique has been adopted in automatic linguistic thesaurus generation (Grefenstette, 1994).

Below, we present some intuitive consideration of why such an improvement is possible. More rigorous proof is presented in the succeeding section.

It is easy to notice that the vast majority of coordinates in the input vectors (documents) are *0*-s, and only very few are *1*-s. So, the typical input vector may look like this:

…00000000010000011000000000001000000100000 1…

This is because in each document most of the terms are missing and only a few are present.

The self-organization algorithm consists of many training cycles, each involving only one input vector. We know that an input vector is typically very sparse. So, the changes in the map resulting from each input are relatively small. To reflect the sparseness present in the input vectors we used special data structures. We represented input vectors as *sets* consisting of all their non-zero coordinates. Each non-zero coordinate was essentially a pair *(x, i)*, where *x* was the coordinate value and *i* specified the axis. If the coordinate was not in the set, its value was equal to *0*. This approach significantly reduced the amount of information to be stored and made processing much faster. The next section shows in detail how it was done.

Our objective was to modify the SOM algorithm so as to be able to compute distances to all nodes (Step 3) and update the weights of nodes (Step 4) at a number of iterations proportional to the number of non-zero coordinates in the input vector, represented here as P. Since we were able to do so, we designed an algorithm that takes $O(PS)$ time instead of $O(NS)$, which may be thousands of times faster with the tasks mentioned in section 3.4. Below, we show how we derived the required modifications.

It should be noted that the modified algorithm produced the same output as the original Kohonen's SOM algorithm.

3.5.4 Mathematical Foundation for the SSOM Algorithm

In this section, we first describe our modifications for weight updates (Step 4 of the SOM algorithm), followed by our procedure for computing distance to all nodes (Step 3).

3.5.4.1 Updating Weights to Nodes

This subsection shows how to update weights at $O(PS)$ time.

We used a special representation for the weights of nodes. Instead of keeping the actual values of weights w_{ij} we introduced so called "scaleable weights" denoted by a_{ij} and a special "scale" factor f_j defined for each node j. The algorithm guarantees that at each training cycle the weights w_{ij} can be computed by the following multiplication:

$$w_{ij}(t) = f_j(t)\, a_{ij}(t) \qquad (3)$$

The algorithm does not store or update w_{ij} , but instead operates with the "scaleable weights" $a_{ij.}$ It starts with $f_j(0)=1$ for all nodes, and $a_{ij}(0) = w_{ij}(0)$ for any i and j. Introducing scale factor f_j benefits the performance because each time the algorithm requires weights w_{ij} to be changed we have a choice between changing a_{ij} or changing f_j. We can always choose the fastest way. Changes in f_j affect values w_{ij} for all coordinates i.

The weight update formula in Step 4 of the SOM algorithm can be transformed into two separate cases:

$$w_{ij}(t+1) = \begin{cases} w_{ij}(t) + h(t)(0 - w_{ij}(t)), \text{ for all } i \text{ such that } x_i(t) = 0 \\ w_{ij}(t) + h(t)(1 - w_{ij}(t)), \text{ for all } i \text{ such that } x_i(t) = 1 \end{cases}$$

and re-arranged into:

$$w_{ij}(t+1) = \begin{cases} (1-h(t))w_{ij}(t), \text{ for all } i \text{ such that } x_i(t) = 0 \\ h(t) + (1-h(t))w_{ij}(t), \text{ for all } i \text{ such that } x_i(t) = 1 \end{cases}$$

In terms of a_{ij}, and f_j, this transformation is expressed as:

$$f_j(t+1)a_{ij}(t+1) = \begin{cases} (1-h(t))\, f_j(t)a_{ij}(t), \text{ for all } i \text{ such that } x_i(t) = 0 \\ h(t) + (1-h(t))\, f_j(t)a_{ij}(t), \text{ for all } i \text{ such that } x_i(t) = 1 \end{cases} \quad (4)$$

We update f_j by the rule:

$$f_j(t+1) = (1-h(t))\, f_j(t) \quad (5)$$

To obtain the update rules for the weights a_{ij}, we substitute the update rule (5) into update rule (4):

$$(1-h(t))\, f_j(t) a_{ij}(t+1) = \begin{cases} (1-h(t))\, f_j(t) a_{ij}(t), \text{ for all } i \text{ such that } x_i(t) = 0 \\ h(t) + (1-h(t))\, f_j(t) a_{ij}(t), \text{ for all } i \text{ such that } x_i(t) = 1 \end{cases}$$

By dividing both sides by $(1-h(t))\, f_j(t)$ we finally obtain:

$$a_{ij}(t+1) = \begin{cases} a_{ij}(t), \text{ for all } i \text{ such that } x_i(t) = 0 \\ \dfrac{h(t)}{(1-h(t))\, f_j(t)} + a_{ij}(t), \text{ for all } i \text{ such that } x_i(t) = 1 \end{cases} \qquad (6)$$

Since the case $x_i(t)$=0 does not change the values for a_{ij} we need to update only the factor f_j and the values a_{ij} corresponding to non-zero coordinates $x_i(t)$ in rule (6). That means the time complexity of our weight adjustment is proportional to the number of non-zero coordinates. Updates according to the above formulas produce the same values for w_{ij} as the traditional SOM algorithm.

3.5.4.2 Computing Distance to All Nodes

Step 3 accounts for the other major part of the computation in SOM. Redoing this process is vital for performance. We follow the same objective of separating zero and non-zero coordinates in the input vector. Throughout our paper, summation $\sum_{x_i(t)=1}$ stands for $\sum_{i=0}^{N-1}$, for all i such that $x_i(t)=1$. Summation $\sum_{x_i(t)=0}$ stands for $\sum_{i=0}^{N-1}$, for all i such that $x_i(t)=0$. It is easy to see that it takes $O(P)$ time to compute the first one, while the second one takes $O(N)$ time. Our objective is to convert all transformations into those that are defined by

$\sum_{x_i(t)=1}$ since they can be computed at *O(P)* time. If we kept any $\sum_{i=0}^{N-1}$ components, our implementation would not be efficient and scalable since *N* can be of the order of thousands and grows as *O(S)*, while *P* is a small constant. Using the above notation, the formula in Step 3 of SOM can be re-written the following way:

$$\sum_{i=0}^{N-1}(x_i(t)-w_{ij}(t))^2 = \sum_{x_i(t)=1}(1-w_{ij}(t))^2 + \sum_{x_i(t)=0} w_{ij}^2(t) = \sum_{x_i(t)=1}(1-2w_{ij}(t)+w_{ij}^2(t)) + \sum_{x_i(t)=0} w_{ij}^2(t) =$$

$$= \sum_{x_i(t)=1}(1) + \sum_{x_i(t)=1} w_{ij}^2(t) + \sum_{x_i(t)=0} w_{ij}^2(t) - 2\sum_{x_i(t)=1} w_{ij}(t) = \sum_{x_i(t)=1}(1) + \sum_{i=0}^{N-1} w_{ij}^2(t) - 2\sum_{x_i(t)=1} w_{ij}(t) =$$

$$= S_1(t) + S_2(t) + S_3(t)$$

Here, $S_1(t) = \sum_{x_i(t)=1}(1)$ and $S_3(t) = -2\sum_{x_i(t)=1} w_{ij}(t)$ can be computed at *O(P)* time. Computing $S_2 = \sum_{All\ x_i} w_{ij}^2(t)$ involves all coordinates and is time consuming. The correct solution is to update it at each cycle instead of computing. We derive the update rule for $S_2(t+1)$ below by using a_{ij} and f_j, as defined in (3) and (6):

$$S_2(t+1) = \sum_{i=0}^{N-1} {w_{ij}}^2(t+1) = \sum_{i=0}^{N-1} (a_{ij}(t+1)\, f_j(t+1))^2 =$$

$$= \sum_{x_i(t+1)=1} (a_{ij}(t+1)\, f_j(t+1))^2 + \sum_{x_i(t+1)=0} (a_{ij}(t+1)\, f_j(t+1))^2 =$$

$$= \sum_{x_i(t+1)=1} (a_{ij}(t+1)\, f_j(t+1))^2 + \sum_{x_i(t+1)=0} (a_{ij}(t)\, f_j(t)(1-h(t)))^2 =$$

$$= \sum_{x_i(t+1)=1} (a_{ij}(t+1)\, f_j(t+1))^2 + (1-h(t))^2 \sum_{x_i(t+1)=0} (a_{ij}(t)\, f_j(t))^2 =$$

$$= \sum_{x_i(t+1)=1} (a_{ij}(t+1)\, f_j(t+1))^2 + (1-h(t))^2 (\sum_{i=0}^{N-1} (a_{ij}(t)\, f_j(t))^2 - \sum_{x_i(t+1)=1} (a_{ij}(t)\, f_j(t))^2) =$$

$$= S_2'(t+1) + (1-h(t))^2 (S_2(t) - S'(t+1)),$$

where $S_2'(t+1) = \sum_{x_i(t+1)=1} (a_{ij}(t+1)\, f_j(t+1))^2$ and $S'(t+1) = \sum_{x_i(t+1)=1} (a_{ij}(t)\, f_j(t))^2$ can both be computed at *O(P)* time. This implies that $S_2(t)$ can be updated at *O(P)* time by using the above rule.

3.5.4.3 What Is the Gain?

So far, we have shown that computing distance to all nodes (Step 3 in SOM) and updating weights of nodes (Step 4) can be implemented in such a way that it takes time proportional to the number of non-zero coordinates in the input vector (represented by *P*), instead of the total number of coordinates (represented by *N*). Because these two operations need to be performed at each iteration, the overall learning time for SOM was drastically reduced while preserving the same output. Our proposed SSOM algorithm is efficient for applications where the average number of features present in a given record

is much lower than the total number of available features in the entire database, i.e., exhibiting the sparse input vector characteristic. Our SSOM algorithm takes O(PS) time instead of $O(S^2)$, so it can be thousands of times faster with up-to-date tasks. From our benchmarking experiments described in the subsequent sections in more detail, we observed that P is almost constantly invariant to the scale S and the resulting time complexity of SSOM is almost linear: $O(S)$.

We have already verified that a similar transformation technique can be applied when the input coordinates are real numbers and not limited to *0* or *1*. All the algorithmic improvements remain the same as long as the input vectors are sparse.

3.6 Benchmarking Tests

3.6.1 Electronic Brainstorming Comment Clustering

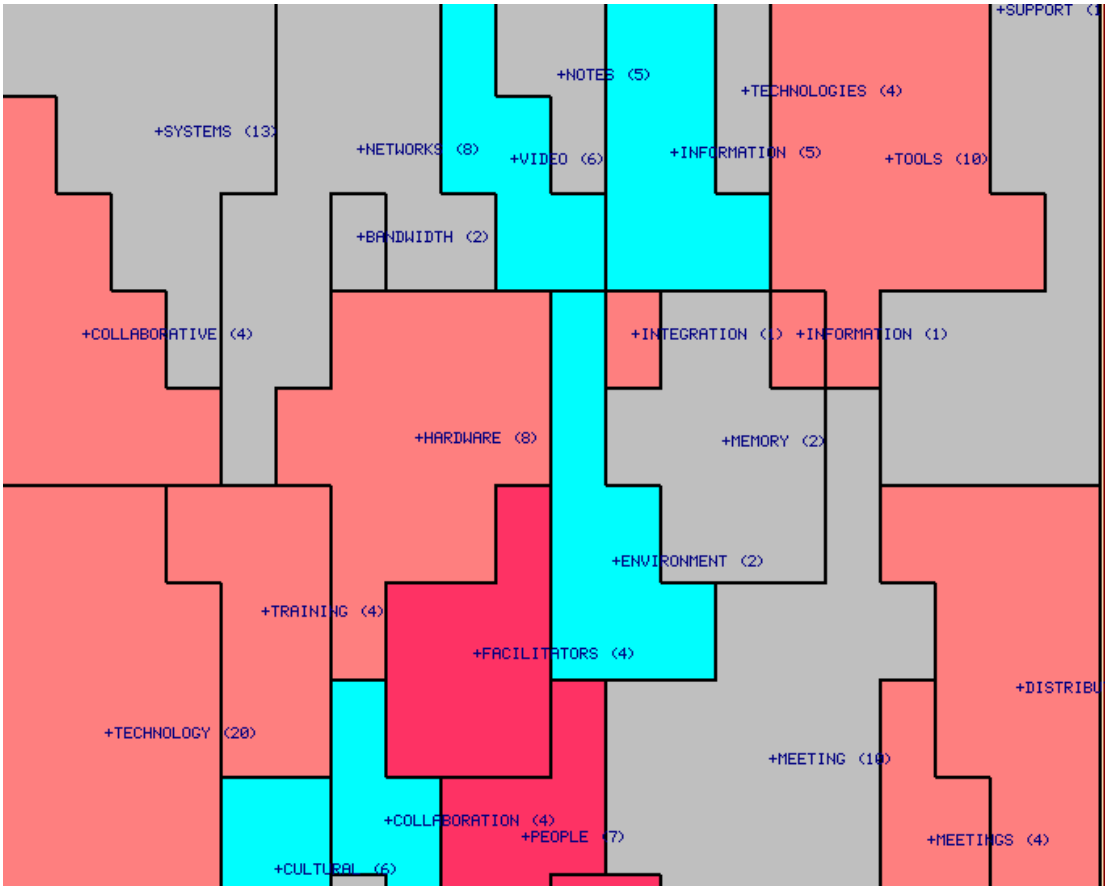


Figure 3.2. SOM for the EBS collection.

Figure 3.2 shows the graphical representation of the map for the EBS collection. It can also be accessed via WWW at: http://ai.bpa.arizona.edu/gsmap/

The output layer consisted of a 20-by-10 grid, (i.e., 200 output nodes). Each input vector varied in size (25-400 terms) and represented the top terms (in frequency) in the input file. Using a DEC Alpha 3000/600 workstation (200 MHz, 128 MBs RAM) for our

experiment, application of the original SOM algorithm took between 55 seconds and 18 minutes, depending on the vector size. As shown in Table 3.1 and Figure 3.3, the SOM processing time increased proportionally to the size of the input vector. However, our SSOM algorithm was invariant to the size of the input vector and was much smaller. The SSOM processing time ranged between 41 seconds and 52 seconds (only small overhead was involved in initializing larger input vectors).

Table 3.1. Meeting comments: processing time as a function of the vector size (number of cycles = 15,000)

VECTOR SIZE	NON-OPTIMIZED (MIN:SEC)	OPTIMIZED (MIN:SEC)	SPEED-UP (TIMES)
25	0:55	0:41	1.34
50	2:06	0:41	3.07
75	3:18	0:42	4.71
100	4:30	0:43	6.28
125	5:41	0:43	7.93
150	6:53	0:43	9.60
175	8:05	0:43	11.28
200	9:16	0:46	12.09
225	10:28	0:45	13.96
250	11:40	0:47	14.89
275	12:51	0:47	16.40
300	14:03	0:48	17.56
325	15:15	0:49	18.67
350	16:26	0:50	19.72
375	17:38	0:51	20.75
400	18:50	0:52	21.73

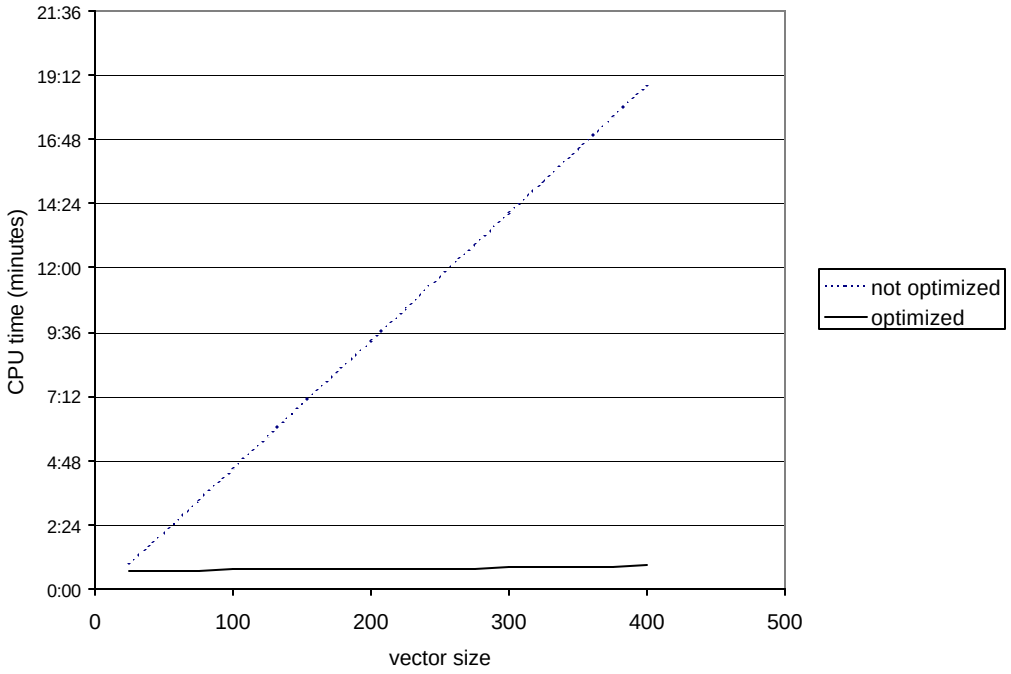


Figure 3.3. Meeting comments: processing time as a function of the vector size

Although this test set appears small in size, the SSOM speed-up (from 1.34 times to 21.73 times) should be considered very important because each EBS session needs to be analyzed and classified in real time during actual meetings (Orwig et al., 1997). The SOM system was often used as an active meeting facilitation agent for classifying textual meeting output (Chen et al., 1996b). A short meeting pause of 1-2 minutes for SSOM generation is considered reasonable. However, waiting for 20 minutes and more for SOM classification in an actual meeting may often disrupt meeting process and dynamics. For longer meeting sessions and larger vector sizes, the SSOM speed-up would be even more significant.

3.6.2 Internet Entertainment Homepage Clustering

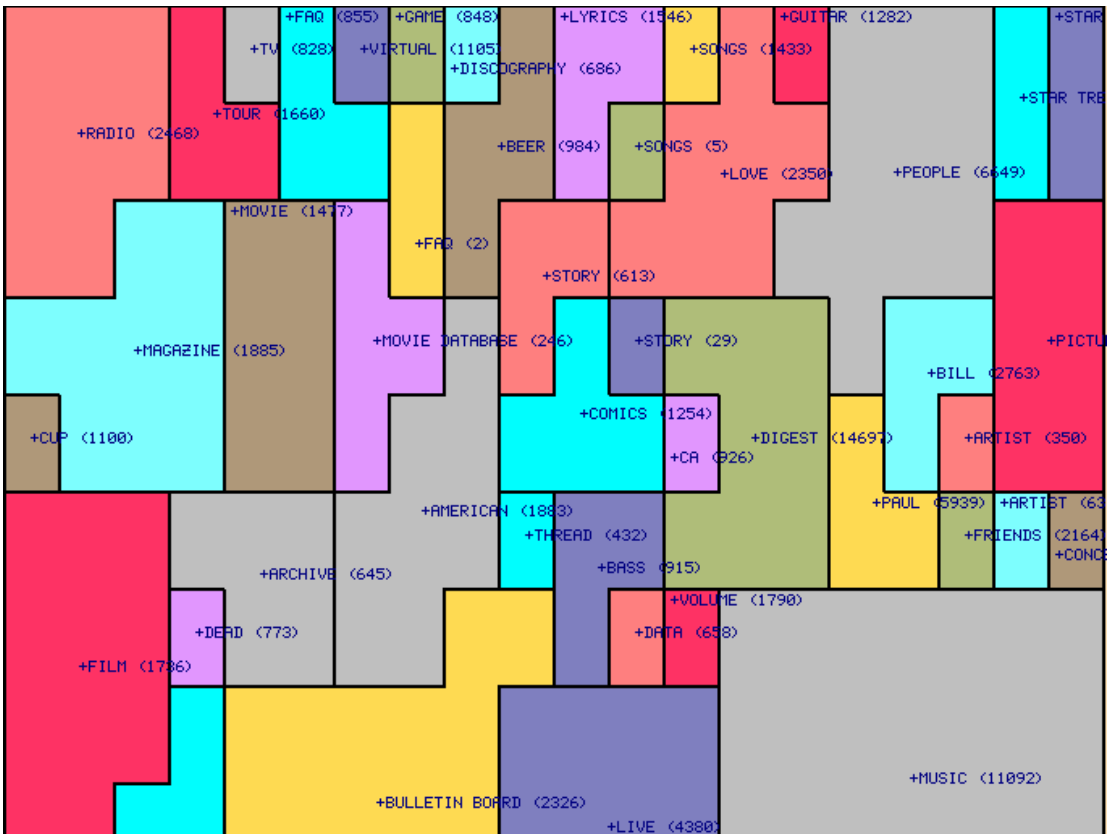
The top-most classification output (entertainment-related topics) represented by SOM is shown in Figure 3.4. Each region represents an important entertainment-related topic (category) determined by SOM and the number of homepages classified under each region was also noted in the map. Each region can be clicked on to view a more detailed SOM map representing sub-topics classified under the region. The threshold for the lowest-level map was set to 100 homepages. By clicking the lowest-level map, a user could view the actual homepages. Readers are referred to (Orwig et al., 1997) for more details. Sample outputs can also be accessed via WWW at: http://ai.bpa.arizona.edu/ent

The same DEC Alpha 3000/600 was used for the SSOM experiment on Internet homepages. Only the first-level map was used for comparison. (Producing maps of all levels took 4-5 times the first-level map processing time.) As shown in Table 3.2 and Figure 3.5, the processing time for the SOM varied significantly according to the input vector size (25 to 400 coordinates), from 10 minutes and 10 seconds to 190 minutes and 20 seconds. Using SSOM, the processing time ranged between 8 minutes and 12 seconds and 12 minutes and 33 seconds. As plotted in Figure 3.5, SSOM was largely invariant to the vector size, while SOM process time was linearly proportional to the vector size. The SOM speed-up was15-fold when vector size was 400.

Due to the time requirement for testing the SOM algorithm, we were unable to continue increasing the vector size beyond 400. We predict that the same speed-up conclusion should hold for large vector sizes. For the large-scale Illinois DLI test collections, which

involve several million Internet homepages and journal abstracts (5-10 GBs) (Schatz & Chen, 1996), significantly larger vector sizes (often on the order of several thousands) are needed, thus demanding the SSOM classification approach.

Figure 3.4. First-level map for 10,000 entertainment-related homepages



vector size	non-optimized (min:sec)	optimized (min:sec)	speed-up (times)
25	10:10	8:12	1.23
50	22:10	8:29	2.61
75	34:11	8:46	3.90
100	46:12	9:04	5.10
125	58:12	9:21	6.22
150	70:13	9:39	7.28
175	82:14	9:56	8.28
200	94:14	10:13	9.22
225	106:15	10:31	10.10
250	118:16	10:48	10.96
275	130:16	11:06	11.74
300	142:17	11:23	12.50
325	154:18	11:40	13.23
350	166:18	11:58	13.90
375	178:19	12:15	14.56
400	190:20	12:33	15.17

Table 3.2. Internet homepages: processing time as a function of the vector size (number of cycles=130,000)

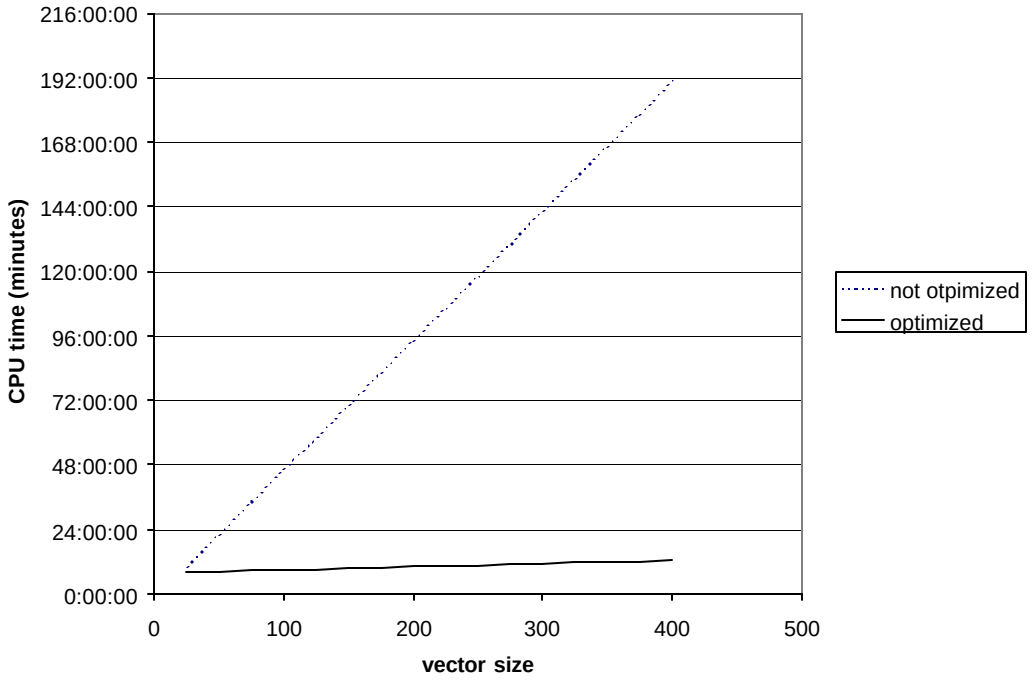


Figure 3.5. Internet homepages: processing time as a function of the vector size

3.6.3 Compendex Abstracts Clustering

We built a map of 30 by 30 nodes and performed 4,500,000 training cycles, so each document was presented about 15 times. The entire process took 15.7 CPU hours, about 24 hours of real time. We used the same DEC Alpha machine for this experiment as for the experiments described above. Our analysis of the algorithms presented earlier in this paper suggests that the straightforward technique would take about 1000 times as much time, which corresponds to 625 CPU days. Even with a supercomputer speed-up ratio of about 40 to 1, this task could never be finished in a reasonable amount of time.

3.7 Conclusions

This chapter has presented our research efforts to develop a scalable and efficient implementation of the Kohonen's self-organizing map (SOM) algorithm. Our data structures and implementation took advantage of the sparsity of coordinates in the document input vectors and reduced the SOM computational complexity by several orders of magnitude. The resulting time complexity of our algorithm is proportional to the average number of non-zero coordinates in an input vector rather than to input vector size, which is the case with the original algorithm.

Through the benchmarking tests, we have found that the actual speed-up ranges from 30% to 20 times, depending on the size of the collection and the chosen input vector size. The implication for small-scale tasks such as clustering electronic brainstorming output is

that SOM can run in real time, taking just a few seconds instead of the half-hour that the straightforward implementation took. In addition, the proposed Scaleable SOM (SSOM) algorithm renders handling large-scale tasks (such as Internet newsgroups or the entire World Wide Web) a possibility. Asymptotic complexity of our algorithm is almost linear: *O(S)*, where S is a generalized input size. Assuming modern CPU speed growth is sustained, we can conclude that the interactive applications of SOM can be extended to larger-scale tasks in the near future.

Our improvements are independent of other possible SOM speed-ups, such as parallel implementations and optimizing the algorithm part that finds the winning node at each iteration. In our future research, we will probably combine them with our approach for better responsiveness and scalability.

Maps larger than 100 x 100 still seem to be rather costly to build, since they require substantial computer memory. Since updating the data associated with the map nodes must be rather frequent, using slow access storage such as magnetic disk or a tape drive would be time prohibitive. Using computer RAM to store these data is so far the only alternative.

The most important implication for the entire dissertation research is that the speed of our algorithm is adequate for real-time interactive applications as long as the number of inputs is less than a thousand (1000), making the algorithm a suitable candidate for clustering and visualizing the top 200-1000 documents in ordered lists of hits returned by search engines from large collections such as Internet.

4. SOM CLUSTERING ABILITIES

4.1 Objectives

As we already have written in the "Literature Review and Research Formulation" chapter, in spite of many successful applications of Kohonen's self-organizing maps in various domains (Kohonen, 1995), its ability to cluster text documents has not been evaluated. In this chapter, we describe our experiment comparing SOM clustering abilities against the very well known Ward's clustering (Ward, 1963) technique. We have based our evaluation on how closely clusters produced by a computer resemble those created by human experts.

For evaluation, we used text messages obtained from group brainstorming meetings (Orwig et al., 1997). This chapter describes our experiment and findings from the perspective of this entire dissertation research. Additional details about the study described in this chapter can be found in (Roussinov & Chen, 1999).

4.2 Research Questions and Methodology

The question answered by this study was: *Can SOM correctly cluster text documents?* Correctness has been measured as concordance between the clusters created by the algorithm and those identified by human experts. We followed empirical methodology and performed a controlled experiment in order to compare SOM against Ward's clustering (Ward, 1963), a very well known technique.

Traditionally in Information Science, clustering techniques have been evaluated in conjunction with retrieval tasks (Hearst, 1996; Croft, 1995). We have found surprisingly few studies involving methodological evaluation of clustering techniques based on resemblance between the resulting partitions and clusters produced by human experts. In addition, we are not aware of any other study involving evaluating document-clustering techniques through experiments with human subjects.

Sahami et al. (1998) based their measurement on whether or not a pair of objects was put into the same class by human experts and by the system. For human expert judgments they used manually created categories existing in the Reuters collection. Zamir et al. (1997) used a similar measurement to test clustering applied to a collection created by merging several smaller collections of Web documents on different topics. We have chosen to use similar metrics in our study. More details follow in the "Experiment Design" section.

4.3 Background and Issues

4.3.1 Electronic Brainstorming Meetings

In this study, we have compared two document clustering techniques using data obtained from electronic meeting sessions described in more detail in (Chen et al., 1994). Efficient organizing of electronic meeting comments is itself a rewarding task. Electronic meeting support has been proven to have great impact on productivity of group discussions (Nunamaker et al., 1991a). Because participants in electronic meetings can generate

hundreds of comments in an hour, the task of categorizing and organizing them is very time consuming. Intelligent agents are believed to be able to significantly reduce the cognitive load of meeting participants (Chen et al., 1994) by automatically organizing documents into clusters, even if manual post-processing still is deemed necessary.

4.3.2 Ward's Clustering

Hierarchical agglomerating clustering (HAC) algorithms are the most commonly used method of document clustering (Willet, 1988). 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 achieved. One of the most popular HAC algorithms is Ward's clustering, proposed by statistician J. Ward (Ward, 1963). Over time, it has been extensively used in various domains: astrophysics, pattern recognition, applied statistics, etc. Ward's clustering has been repeatedly applied for text analysis; El-Hamdouchi & Willett (1986) is an example.

Murtagh (1985) has proposed the reciprocal nearest neighbor approach (RNN) which is significantly faster than the straightforward implementation but produces identical results. The advantage is a resulting time complexity of $O(N^2)$ in comparison with $O(N^3)$ for the classical implementation. N represents the number of inputs, which in our case are meeting comments.

4.3.3 Statistical vs. Neural

Although, both techniques may serve to cluster data, they do it in different ways. Statistical techniques proceed by pair-wise comparison of objects. Neural networks proceed by a process called *learning*. Neural networks are believed to possess some particularly valuable properties, since they are patterned after associative neural properties of the brain. An additional rationale for our research was that it would be useful to compare the techniques of these two very different approaches and see which performed better in the domain of unstructured text.

4.4 Testbed

We used the output from an electronic brainstorming meeting (Chen et al., 1994) containing 206 comments. The meeting participants discussed the issue "The Future of GroupWare."

We treated each textual comment entered by a meeting participant as an independent document. EBS comments exhibit some unique characteristics and often contain typos, abbreviations, and incomplete sentences. Typically, a one-hour session with a dozen or so participants generates several hundred comments. Since manual clustering of the entire session would be very time consuming, we asked a human expert to choose 80 comments falling into approximately 8-10 categories (topics) and used only the selected comments in our experiment.

4.5 Algorithms and Implementations

This section explains in general terms how we implemented our automatic text clustering systems. Our process proceeds by the following steps:

1. Automatic indexing.
2. Selecting most discriminating terms for document representation.
3. Applying clustering technique: SOM or Ward's.

4.5.1 Automatic Indexing

The purpose of automatic indexing is to identify the content of each textual document automatically by sets of associated features (Salton, 1983). Features are words and phrases. Automatic indexing first extracts a set of all words and possible phrases that enter the document. Then it removes words from a ``stop-word" list to eliminate non-semantic bearing words such as `the, a, on`, and `in`. Our automatic indexing program also creates phrases from adjacent words. In this research, we used the automatic

Table 4.1. 20 most frequently occurring terms in the collection.

0 MEETINGS	10 NETWORKS
1 MEETING	11 SUPPORT
2 TECHNOLOGY	12 NOTES
3 COLLABORATIVE SYSTEMS	13 HARDWARE
4 INFORMATION	14 FACILITATORS
5 COLLABORATIVE	15 TECHNOLOGIES
6 DISTRIBUTED	16 LANGUAGE
7 SYSTEMS	17 WIRELESS
8 LINEAR THREAD MEETING	18 NETWORK
9 ENVIRONMENTS	19 BANDWIDTH

```
Form a cluster from each document.
Until only one cluster remains do
 Pick arbitrary cluster as Current Cluster;
 Found = False;
 Until not Found do
  find the closest neighbor to Current Cluster;
  if they are reciprocal nearest neighbors then
      Merge them;
      Found = True;
  otherwise
      Change Current Cluster to its nearest neighbor;
 end-do
end-do
```

Figure 4.1. The pseudocode for Ward's clustering.

indexing described in (Orwig et al., 1997). We already have described the indexing process on page 51.

4.5.2 Document Representation

For computational efficiency and accuracy of representation, we preserved only the top 100 most frequently included terms in the collection. This approach works best with small collections consisting of short text messages, since it provides the greatest overlap in representations. Table 4.1 shows a list of the 20 most frequently appearing terms in the collection that we used. The average number of keywords preserved in the representation of a document was 3.7.

We used the Information Science community's most popular representation of documents: vectors in vector space (Salton, 1989). Each coordinate in the vector space corresponds to a term. A term can be a single word or a phrase. In our case, if a term did not enter the document, the corresponding coordinate was set to 0. If a term entered the document, we set the coordinate to 1. Prior research (Orwig et al., 1997) has shown this scheme to be adequate for electronic meeting messages.

4.5.3 Ward's Clustering Implementation Issues

4.5.3.1 Speed

In our study we also used the RNN approach (Murtagh, 1985), described in pseudocode in Figure 4.1. By definition, two clusters, C1 and C2, are called reciprocal nearest neighbors if C1 is the nearest neighbor for C2, and C2 is the nearest neighbor for C1. The algorithm uses the inverse Euclidean distance between the centroids of clusters as the measure of similarity between those clusters. The centroid of a cluster is computed by averaging the coordinates of all documents in the cluster. Murtagh (1985) gave a proof that this algorithm terminates and produces output that does not depend on the order of selecting a cluster as the current cluster.

We have also found another modification crucial to making the approach computationally tractable and scalable. In the text analysis domain, the dimensions of the vector space are large. In this study the input vector size was 100. If we represent the vector size by N, the time complexity of the straightforward similarity computation is $O(N)$, because Euclidean

distance requires iterations through all the coordinates. Our modification requires only non-zero coordinates in a vector representation of documents to be stored, for example as a linked list. When distance is computed, the iteration cycle is organized in such a way that it goes through only non-zero coordinates. This changes the *O(N)* complexity into *O(M)*, where *M* is the average number of non-zero coordinates in the document representation, which can be hundreds of times smaller than *N*.

4.5.3.2 Dendrograms and Partitions

Ward's algorithm runs until all documents are merged into a single cluster containing all of them and produces a balanced binary tree called a *dendrogram*. Each node in the dendrogram corresponds to a cluster obtained as a result of merging two other clusters corresponding to the child nodes. Since in our study we evaluated partitions, we converted the dendrogram into a partition. Murtagh (1985) suggested a variance threshold technique for this purpose. The technique traverses the dendrogram tree and splits clusters associated with a node into two clusters associated with the corresponding child nodes. It stops when the average similarity between documents in each cluster and its centroid exceeds the specified threshold.

We found this approach to be inadequate for using Ward's clustering in our task. The variance threshold approach often produced one big cluster, containing half of all the documents not necessarily similar to each other, and many small clusters, with 1-3 documents in each. We devised a so-called *shared keyword rule* to alleviate the above problem. It followed the same recursive traversal procedure as the variance threshold but

split the clusters only if there was no common keyword entering all documents in the cluster.

In more detail, the algorithm starts from the root of the dendrogram that Ward's clustering produces. The algorithm checks for the presence of a term (word or phrase) that all the comments below the root have. If not, the algorithm assigns all the documents to two clusters according to the dendrogram. It then recursively checks each of the two obtained clusters in the same way. This way, the dendrogram influences only decisions on how to split the clusters, but not when to stop. At the end, documents in each cluster have at least one term in common. We adopted this approach because we followed a similar approach in the implementation of SOM: The regions in the SOM were formed by merging map nodes that had the same most representative terms (Orwig et al., 1997). We empirically found that this implementation of Ward's clustering resulted in a greater number of clusters of meaningful size (three or more documents). Since the comments in each cluster had at least one term in common, they were also likely to discuss similar issues.

4.5.4 SOM Implementation Issues

Figure 4.2 shows the map that we generated and used in the current research. Each region represents a topic (or cluster). The number following each topic represents the number of documents assigned to each topic. Since in this study we evaluated only clustering properties, we converted the produced map into a partition before conducting

experiments. The human subjects did not see either the map itself or the labels for the regions.

4.6 Experiment Design

4.6.1 Procedure and Assumptions

Our experiment involved 17 human subjects, volunteering business school students (10 graduate and 7 undergraduate) familiar with the topic of discussion and not involved in this research.

Each subject received two text files that contained text comments grouped into topics (clusters). Subjects were asked to re-arrange the comments in the file according to their own judgment. One file was the output from SOM; the other was the output from Ward's clustering. The order of file presentation was reversed for half of the subjects. It took 40-50 minutes on average for a subject to perform the task. Subjects worked on their own. We did not try to observe how subjects moved the comments around.

*** New Topic ***

Currently collaborative rooms are expensive to set up and maintain.
We need to be able to do portable environments in settings where we
would have normal meetings. This would entail the use of wireless lan
technology that work beyond the line of site in the current technology

Cellular technology hold great promise for having remote
Collaborative meetings but the reliability and band with need to be
Improved.

The MOST important is to somehow make sure that the technology does
Not overtake the human part of human interaction.

User awareness of the capabilities and benefits of using this
technology----"Getting the word out".

Repository technology will have to move toward an acceptable
Standard that will promote optimal sharing between various types of
Environments. Don't make our own standards.

*** New Topic ***

Understanding how our technology encapsulates specific cultural
Expectations -- and learning which technologies are appropriate in
Which cultures.

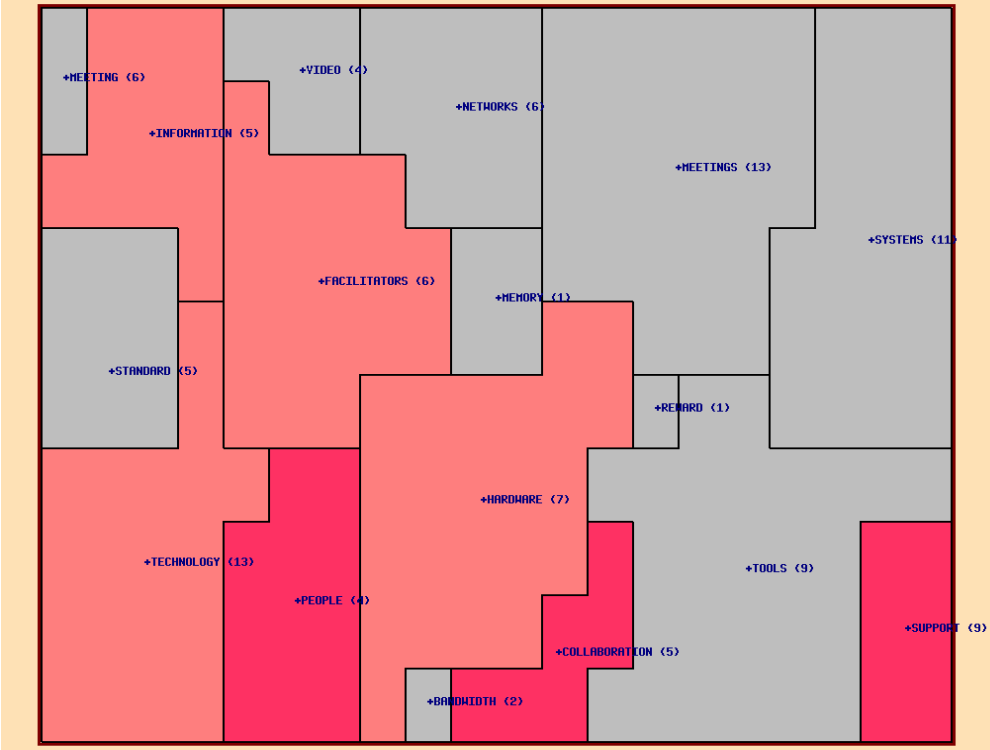
How to measure the validity and reliability of groupware tools

We need to build "techniques" or "processes" on top of the raw
tools we are currently developing for electronic group settings.

Figure 4.3. Sample EBS comments

It should be noted that in this design subjects were given an initial set of clusters created by a computer, not text comments in random order. The final partitions created by the

Figure 4.2. The self-organizing map that we used for our study.



subjects therefore may have been influenced by our initial partitions. We deliberately designed our experiment in this way since we were interested in the amount of effort that it would take to change the partition suggested by a computer to one the user deemed adequate. This sequence simulates the semi-automatic classification of text collection that takes place in electronic meetings and other tasks where classification of text documents involves some additional manual work.

Figure 4.3 shows a portion of one of the files given to the subjects. Comments follow each other sequentially in a file from top to bottom, separated by at least one empty line. A special text string ("*** New Topic ***") separates clusters from each other. Clusters did not have any labels associated with them. Since the outputs fom both clustering algorithms looked structurally the same, the subjects were unable to distinguish between the techniques involved.

4.6.2 Metrics

To measure the quality of clusters we used their "closeness" to the clusters created by humans. We based our metrics on the time that the subjects spent "cleaning up" clusters created by a computer and the number of wrong and missed associations. The definitions below help to explain these metrics.

We call a partition created by an expert a *manual partition*. An *automatic partition* is one created by a computer. Inside any partition, an *association* is a pair of documents belonging to the same cluster. *Incorrect associations* are those that exist in an automatic

partition but do not exist in a manual partition. *Missed associations* are those that exist in the manual partition but do not exist in an automatic partition. We define *clustering error* as:

$$CE = \frac{E}{P_t}$$

where P_t is the total number of possible pairs of documents: P_t = ½ D (D - 1). E represents the total number of incorrect and missed associations: $E = E_i + E_m$.

This measure favors small partitions. To provide less dependence on the size of both partitions, we also used a *normalized clustering error*, expressed as:

$$NCE = \frac{E}{A_t}$$

Here, A_t is the total number of all associations in both partitions without removal of duplicates (associations existing in both partitions). It is computed as $A_t = A_m + A_a$, where A_m is the total number of associations in the manual partition and A_a is the total number of associations in the automatic partition. We considered only associations from clusters representing three or more documents. It is easy to verify that this measure belongs to a [0, 1] interval.

We also adopted *cluster recall* and *cluster precision* similarly to the measures of *recall* and *precision* typically used in information science research (Salton, 1983). Rather than

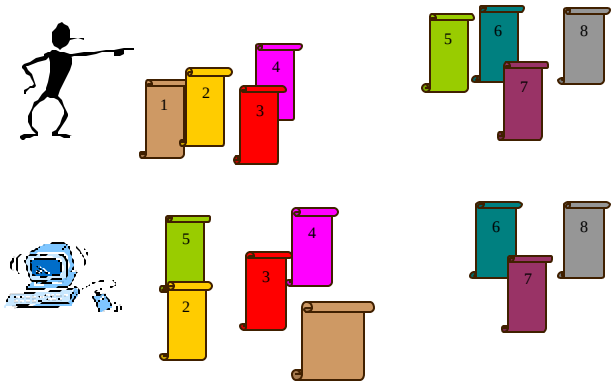


Figure 4.4. The example of computing clustering error, recall, and precision.

examining the number of relevant documents, we counted the number of correct associations. Therefore, we define cluster recall as:

$$CR = \frac{A_c}{A_m},$$

where $A_c = A_a - E_i$ represents total number of correct associations in automatic partition. We define cluster precision:

$$CP = \frac{A_c}{A_a}$$

It is easy to see that cluster recall reflects how well the clustering technique detects associations between documents and that cluster precision reflects how accurate the detected associations are.

Figure 4.4 shows an example of manual partition (above) and automatic partition (below). In this example, the clustering algorithm made a mistake by placing document 5 with documents 1, 2, 3, 4 instead of 6, 7, 8. The incorrect associations are 5-1, 5-2, 5-3,

5-4. The missing associations are 5-6, 5-7, 5-8. The clustering error is 7/ (1/2*8*(8-1)) = .25. The normalized clustering error is 7 / (6 + 6 + 10 + 3) = .28. Cluster recall is (10 + 3 – 4) / (6 + 6) = .75. Cluster precision is (10 + 3 – 4) / (10 + 3) = 0.69.

4.6.3 Research Questions Operationalized

Using the metrics introduced above, we formulated our research questions as follows:

Q1: Which technique produces output that requires less time for an expert to produce final clusters?

Q2: Which technique provides greater cluster recall?

Q3: Which technique provides greater cluster precision?

Q4: Which technique provides smaller clustering error?

Q5: Which technique provides smaller normalized clustering error?

4.7 Results and Discussion

Subject	SOM	Subject	Ward
1	45	1	10
2	90	2	10
3	20	3	15
4	60	4	90
5	50	5	60
6	29	6	60
7	27	7	60
8	90	8	35
		9	40
average	51		42
standard deviation	27		28
confidence interval	31		18
standard mean error	9.6		9.2

Table 4.2. Time spent on the tasks (in minutes).

Subject	Ward	SOM
1	0.11	0.13
2	0.48	0.85
3	0.23	0.25
4	0.26	0.16
5	0.37	0.13
6	0.47	0.26
7	0.25	0.75
8	0.26	0.21
9	0.45	0.18
10	0.33	0.16
11	0.09	0.09
12	0.21	0.26
13	0.14	0.21
14	0.22	0.14
15	0.12	0.12
16	0.06	0.07
17	0.22	0.29
average	0.254	0.254
standard deviation	0.13	0.22
confidence interval	0.063	0.10
standard mean error	0.032	0.052

Table 4.3. Cluster Recall.

Below, we summarize our experimental findings. For all statistical significance tests, we used paired t-tests because two measures produced by the same subject were not independent. The only exception to this was time spent on the task, since we used only the first measurement from each subject. The data collected from the subjects, along with the statistical results, are presented in Tables 4.2-4.6.

Q1. The subjects spent less time correcting the Ward's clustering results than correcting the SOM results. However, the difference was statistically insignificant. The mean times spent on the task were 51 minutes for SOM and 42 minutes for Ward's clustering. The p-value was 0.19. Table 4.2 shows time spent on the task for all subjects. Since the variance in both groups was large, larger sample size appears to be necessary in order to establish statistical significance.

Q2. There was no statistical difference between the SOM results and the Ward's clustering results in cluster recall (Table 4.3). The mean cluster recalls for both SOM and Ward's clustering were 0.25. The p-value was 0.30. Both techniques performed equally in detecting associations between documents. The 95% confidence interval for the difference in cluster recall was established at 0 ± 0.09. This is a rather wide interval; a larger sample size might establish a statistically significant difference.

Q3. Ward's clustering produced significantly better cluster precision than SOM (Table 4.4). The mean cluster precisions were 0.38 for SOM, and 0.69 for Ward's clustering. The p-value for the paired t-test was 0.0014. The implication of this result is that Ward's clustering was more accurate in establishing associations between documents.

Subject	Ward	SOM
1	0.54	0.34
2	0.90	0.82
3	0.71	0.38
4	0.87	0.29
5	0.62	0.17
6	0.27	0.17
7	0.76	0.77
8	0.78	0.32
9	1.00	0.35
10	1.00	0.32
11	0.62	0.44
12	0.66	0.26
13	0.46	0.33
14	0.75	0.37
15	0.81	0.48
16	0.41	0.28
17	0.54	0.28
average	0.70	0.38
standard deviation	0.20	0.18
confidence interval	0.097	0.084
standard mean	0.050	0.043

Table 4.4. Cluster Precision.

Subject	Ward	SOM
1	0.17	0.020
2	0.034	0.022
3	0.082	0.108
4	0.083	0.15
5	0.048	0.14
6	0.034	0.092
7	0.084	0.032
8	0.078	0.13
9	0.040	0.15
10	0.064	0.17
11	0.22	0.36
12	0.074	0.12
13	0.11	0.13
14	0.094	0.19
15	0.24	0.30
16	0.20	0.28
17	0.076	0.096
average	0.104	0.160
standard deviation	0.132	0.180
confidence interval	0.032	0.044
standard mean error	0.016	0.022

Table 4.5. Clustering Error.

Q4. The SOM results exhibited significantly higher clustering error than the Ward's clustering results. The mean clustering errors were 0.080 for SOM and 0.051 for Ward's clustering. The p-value was 0.001. Table 4.5 shows the measurements for all subjects. However, since the clustering error measure favors a technique producing fewer clusters, this is still not an ideal indication of quality. A more objective measure is considered in the next paragraph.

Q5. Ward’s clustering produced a lower normalized clustering error (Table 4.6) than SOM. The mean normalized clustering errors were 0.71 for SOM, and 0.64 for Ward’s clustering. The p-value for the paired t-test was 0.08. The implication is that, over all, the partition produced by Ward’s clustering was closer to partitions produced by human experts.

We randomized the order of tasks. In addition, in order to establish whether the order of tasks influenced the measures, we ran a regression of each measure on the order of tasks represented as a boolean variable. We assigned 0 to this variable if Ward's clustering was processed by the subject first, and 1 if second. We did not find any statistically significant dependency. The minimum p-value for the coefficient in the regression was 0.20 in the case of normalized clustering error. This established that the influence of the order of tasks had been extremely weak and was not statistically significant. This was in agreement with the fact that most subjects did not even notice that the text messages were the same, but in the different order. Since the task was very laborious nobody appeared to have enough patience to verify that.

Table 4.6. Normalized Clustering Error.

Subject	Ward	SOM
1	0.82	0.81
2	0.37	0.16
3	0.60	0.61
4	0.59	0.78
5	0.53	0.85
6	0.62	0.23
7	0.68	0.71
8	0.60	0.74
9	0.37	0.75
10	0.49	0.78
11	0.84	0.85
12	0.62	0.76
13	0.78	0.74
14	0.65	0.78
15	0.78	0.80
16	0.89	0.89
17	0.65	0.79
average	0.64	0.71
standard deviation	0.15	0.20
confidence interval	0.072	0.096
standard mean error	0.037	0.045

We observed that the clusters created by subjects did not always share common words. This is not surprising, since subjects clustered the comments based on their meaning but not the keyword representation as the algorithms did. For example, the comment “Effective transmission of video over networks” was placed by a subject into the same cluster with the comment “bandwidth concerns -- impact of remote collaboration” presumably since both relate to networking issues. Those two comments do not have any common words, so they would never be placed together by our implementation of Ward’s

clustering and are quite unlikely to be so placed by SOM. This helps explain discrepancies between automatic and manual partitions.

4.8 Conclusions

The overall result is that on the EBS data set that we used Ward's clustering performed better. In addition, it produced a smaller number of associations, but was more accurate. The accuracy may be due to the "shared keyword rule" that we implemented, requiring the documents in a cluster to have at least one keyword in common. This conclusion applies only to the particular type of collection we used (electronic meeting messages) and may be sensitive to collection size. Since we are not aware of any evaluation study that was based on manual categorization of the output of automatic categorization and performed on a larger scale, we believe the results of our small-scale experiments are valuable. This is also the first empirical study in the domain of text analysis of Kohonen self-organizing maps as a clustering tool.

Our research also resolved implementation issues related to two automatic text clustering techniques: Ward's clustering and Kohonen's Self-organizing Maps. These issues include:

1. Finding appropriate document representation for this task.
2. Adapting both techniques for creating non-overlapping partitions of text documents.
3. Resolving scalability issues by taking advantage of sparseness of representation in the domain.

We have concluded that our implementation of Ward's clustering is slightly more precise in detecting associations between documents, but that the performances of these techniques in terms of recall of those associations are not statistically different. This suggests that Kohonen's self-organizing map has clustering abilities close to those of known clustering techniques. Since the implementation of SOM for text analysis offers several additional valuable features such as providing labels for and visualizing proximity of the clusters, it is a viable option for interactive search system. The final evaluation reported in "SOM and Internet Search" chapter also indirectly confirms good SOM clustering abilities by demonstrating the overall system effectiveness.

5. CUSTOMIZABLE SOM

5.1 Objectives

As we wrote in our "Literature Review and Research Formulation" chapter, automatically created maps of concepts give a high-level picture about a collection of text documents. However, the output of unsupervised text processing algorithms, including the self-organizing map, does not always match user's expectations (Chen et al., 1996; Hearst, 1999). As a result, it is not yet known whether the interactive information access based on unsupervised text clustering can reduce information overload.

This chapter describes our exploration of potential solutions to the quality problems associated with unsupervised techniques. Although we specifically addressed Kohonen's self-organizing maps, we believe that our approach generalizes to other tools that utilize the idea of visualizing collections of text documents through clusters of documents represented as concept maps, such as SEMIO (www.semio.com) or "Live Topics" (www.altavista.com).

This chapter also explains how we proceeded with building a prototype that allows searching the entire World Wide Web. From the study described in this chapter, we have learned many lessons and made many changes that resulted in creating a system used in the study evaluating the overall approach presented in the next chapter. We had to overcome the myriad technical challenges involved in building a system that allows

interactive Internet search. This chapter is a more detailed compilation of studies reported in (Roussinov, 1999; Roussinov & Ramsey, 1998).

5.2 Research Questions and Methodology

The focus of this study has been *the features that may overcome SOM quality problems reported in prior research.* We have suggested improving the technique by adding a customization component. By *customization*, we mean enabling a user to modify the way in which the system visualizes a collection of documents. Contrary to the systems mentioned in the "Literature Review and Research Formulation" chapter, our approach allows the user to customize the way in that system performs visualization of text collections, rather than being restricted to the system's settings, usually the same for all users and tasks. Through interaction, the system tries to offer concepts that are more meaningful to the user, thereby making it more likely for the user to use the concept map to reformulate the task or navigate the returned documents.

Since we suggested a potential solution through customization, we faced additional research questions: *Can a user understand customization features? Can customization features help?*

To answer these study research questions, we followed the System Building Methodology (Nunamaker et al., 1991) already mentioned in chapter 3. For this purpose, we created and tested a prototype Customizable Self-organizing Map (CSOM) system that builds and refines in real-time a map of concepts found in Web documents returned

by a commercial Internet search engine. Based on a refined map, the user is able to navigate retrieved documents or reformulate the query for more precise search results.

We performed a user study based on two search sessions. We helped subjects perform searches and asked them to "think aloud" to express their reasoning and opinions. We also analyzed the usage patterns that we obtained from the trace files of 50 other search sessions. Because our prototype has been available on the Web for demonstration purposes and attracted some attention from Web surfers, our Web server has automatically recorded their interaction with our system. Both the protocol analysis performed and statistics obtained from usage patterns have provided indirect evidence of success of the suggested approach.

5.3 Background and Issues

Many researchers and practitioners consider Internet searching one of the most challenging areas for future research involving National Information Infrastructure (NII) applications (Bowman, 1994; DeBra & Post, 1994; Pinkerton, 1994). Recently, “Science Daily Magazine” wrote about the web (Lawrence, 1999):

“The web is transforming society, and the search engines are an important part of the process. Consumers use search engines to locate and buy goods or to research many decisions (such as choosing a vacation destination, medical treatment or election vote) Search engine indexing and ranking may have economic, social, political, and scientific effects. For example, indexing and ranking of online stores can substantially effect

economic viability; delayed indexing of scientific research can lead to the duplication of work or slower progress; and delayed or biased indexing may affect social or political decisions."

As we already have written, the problem of information overload has become especially pressing on the Internet, where the prevailing information retrieval mechanisms provided by WWW software are based on either keyword search or directory browsing. Keyword search engines return thousands of web pages for each single information request. Most users go no farther than the first 10-30 documents before quitting in order to reformulate a query or simply give up. The bulk of potentially relevant information remains unexplored. Clustering and summarization tools described in this dissertation help to go beyond those top 10-30 documents by undertaking to understand and modify the search results.

It has been noted (Hearst, 1997) that most Internet users are in general computer novices and in particular are unfamiliar with search technology. The Internet has removed the intermediary role of a librarian, typically present in standard library-science information seeking models (Ingwersen, 1994).

5.4 Testbed

We chose Internet as our testbed for the study reported in this chapter. The rationale behind this decision is based on concern for the increasing importance of the Internet for E-commerce as its size surpasses that of any other publicly available collection of

Table 5.1 Adaptive Features.

Command	Adaptation	Problems addressed
Remove this term	Removes the term from the vector space representation of all documents.	irrelevant terms, unfamiliar terms, meaningless terms
Use more specific terms	Changes statistical thresholds for term selection for the vector space representation of all documents.	Terms not carrying any additional information about the collection.
Use more general terms	Also changes statistical thresholds for term selection for vector space representation of all documents.	Unfamiliar terms, too narrow terms
Create fewer concepts	Uses a smaller grid for the map	Not fully converged maps, too many concepts on the map
Create more concepts	Uses a bigger grid for the map.	Too few concepts on the map

documents and popularity of Internet formats for company intranets (Ba et al., 1997; Davenport & Prusak, 1998; Nonaka, 1994). The preceding section reviewed Internet peculiarities.

5.5 Prototype Design

5.5.1 Rationale behind the Approach

In their user study, Chen et al. (1996) have identified the following problems that sometimes surface with the applications of graphical self-organizing concept maps:

1. Irrelevant, unfamiliar to the user, or meaningless concepts.

2. Concepts of different level of detail at the same level of hierarchy.

3. Too many concepts on the map, or too few.

We have suggested overcoming those problems through features supporting customization of those maps. Table 5.1 contains a summary of features and the problems that those features address. We believe that through interaction, the system tries to offer categories that are more meaningful to the user. Thereby, the user is more likely to use the concept map to reformulate the task or navigate the returned documents.

Rich interaction with the system also introduces an element of entertainment into the search process, which is believed to be beneficial to an information system (Carroll et al., 1988). Removing unwanted concepts is comparable to shooting dragons in an arcade game.

Below, we present more details about the features suggested for customizing self-organizing maps.

5.5.2 Features

A demonstration of CSOM is available on the Web at:

http://ai.bpa.arizona.edu/resume/dmitri/query.html

The search process with CSOM consists of two general steps: 1) refining the concept map without changing the set of visualized documents; and 2) using the concept map to navigate the retrieved documents or reformulate the query. Step 1 is not present in currently available search systems.

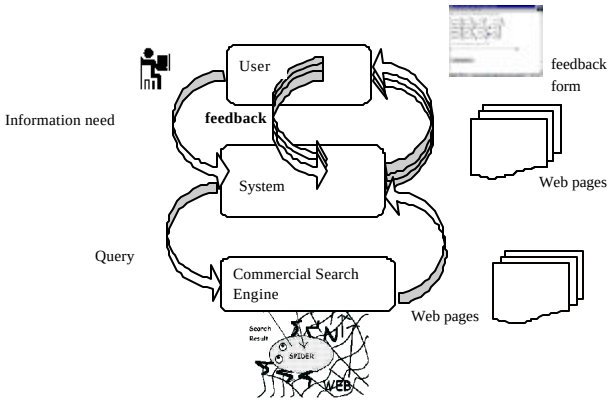


Figure 5.1. User/system/search engine interaction.

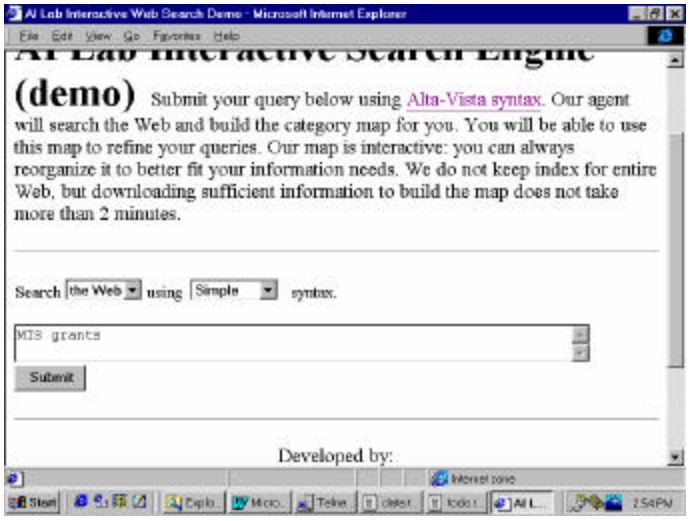


Figure 5.2. Entering the query "MIS grants."

Figure 5.1 shows the user/system/search engine interaction. Section 5.6.1 gives examples of search sessions. The search follows these steps:

1. The user enters a query on an HTML form (Figure 5.2).

2. Our system, CSOM, implemented as a CGI script, routes the query to an external search engine. We chose AltaVista because current evaluations by popular magazines (www.searchenginewatch.com) tout it as one the most popular and powerful Internet search engine.

3. The search engine returns a ranked list of URLs and their short summaries (called snippets) to CSOM.

4. CSOM parses the summaries (or full content on demand) of the top 200 pages on the list, which in our tests typically takes 1-2 minutes.

5. The system builds a self-organizing map and invokes a Java applet for display. The map shows regions (clusters of documents), laid out on a 7 by 7 grid, each colored differently and labeled with a word or phrase. This building process typically takes 3-6 seconds.

6. The user may then interact with the map as follows:

a) *Modify* the query and thus obtain another set of documents and their map;

b) *Inspect* any region (cluster) by browsing the list of web pages mapped into that region;

d) *Customize* the map to better understand the collection of found pages. The most important customizing commands are the following:

"Remove this term." The system removes the specified term from representation of the pages and rebuilds the entire map.

"Please be more specific." The system rebuilds the whole map and creates concepts based on phrases instead of single word terms.

"Fewer or more concepts." The system creates a smaller or bigger map, by using the same list of terms to generate fewer or more concepts."

"Re-do it! I don't like anything." The system replaces all the parameters that were used to create the map with arbitrary parameters to generate a fresh perspective.

Each time the user rebuilds the map takes about 3-6 seconds. Subsection 5.6 contains more examples of invoking the features described above.

5.5.3 Technical Challenges

This section offers an overview of the technical challenges we faced while developing our prototype. Most of them originated in the rather chaotic evolution of Internet applications standards and lack of maturity in Internet technologies. The Internet started as a hypertext platform, not intended for sophisticated graphical user interfaces. So far, we see very few applications on the Web supporting interfaces conforming to the

standards of single-computer (non distributed) operating systems such as Windows-95 or X-Windows.

Most Internet applications are still based on the CGI script “click and wait” paradigm, where the server side application reacts to user requests for web pages located on its server. This is a very limited paradigm if compared with, for example, the dynamic binding of variables and their GUI presentation supported by MS Windows API.

Our prototype has been implemented through integration of modules written in different programming languages: C/C++ and Unix scripts for the server side, Java for displaying concept maps, and HTML forms for query input. We discovered that the behavior of Java applets is specific to the browser (Internet Explorer, Netscape, Mosaic), which it was not the intention of designers of the language.

Another challenge has been interacting with commercial search engines. At present, they do not provide any application interfaces (APIs), so we had to follow heuristic solutions. Our server script pretends to be a human user and asks commercial search engines for particular information by composing special URLs and calling Java applications to fetch from the Web the pages specified by these URLs. For example, getting the page

http://www.altavista.digital.com/cgi-bin/query?pg=q&kl=en&stq=0&q=information+retrieval

from the Web would cause AltaVista to perform a search for a query “information retrieval” and return the result in the form of HTML pages. Our server parses those results and extracts information about the web documents found by AltaVista.

The work reported in this dissertation is based on many modules developed in the AI laboratory in prior research. This includes automatic indexing, vector space representation of text documents, building Kohonen's self-organizing maps and displaying them using Java applets. It has required significant effort to integrate them and adapt them to this particular purpose.

5.6 User Study

5.6.1 Two Search Sessions Observed

We performed a pilot study based on two search sessions in order to see if the proposed approach is viable. Below, we present those cases. We helped subjects to perform their search sessions and asked them to "think aloud" to express their reasoning and opinions.

5.6.1.1 "MIS Grants" Task

Our first subject was interested in grants available to MIS researchers. She first tried the Alta-Vista search engine and entered "MIS grants" as a simple query. She received a page informing her that 36952 documents were potentially relevant. The user was not patient enough to go through many of them. She checked the first 20, judged most of them irrelevant to her search objective and gave up using the search engine directly.

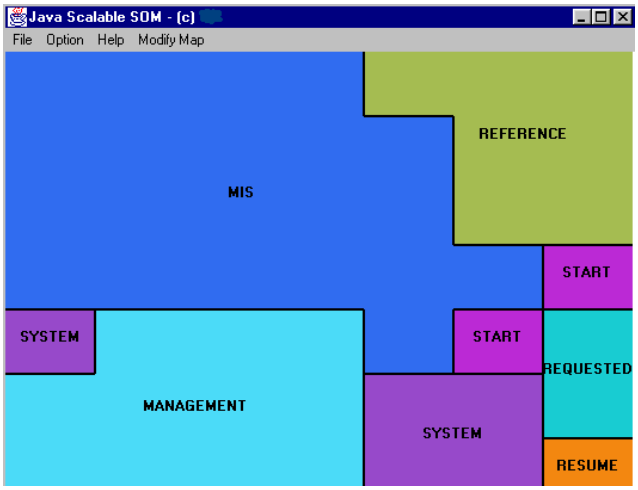


Figure 5.3. The concept map for the query "MIS grants."

Then she resorted to CSOM. She entered the same query in a simple HTML form: "MIS grants." CSOM built a map for the first 200 documents in 40 seconds (Figure 5.3).

The user noted that the terms *mis*, *management*, and *data* looked relevant to her topic of interest. But, she said, they also did not help to focus on relevant documents, since in this particular context they did not convey any additional meaning. Since the query was about MIS grants, terms like *MIS* and *grants* looked trivial and not very helpful. All documents were likely to be about MIS and grants anyway. She removed those terms from the map by activating a local menu and choosing the "Remove this term" menu item.

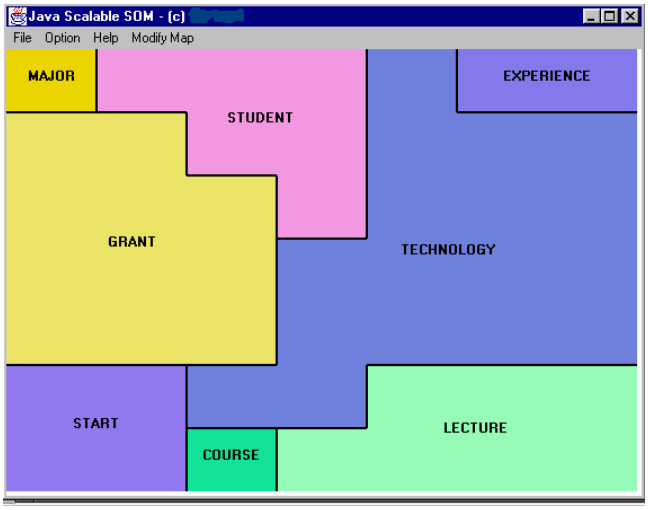


Figure 5.4. The concept map for the query "MIS grants" after the user removed the concepts "MIS," and "grants."

The user judged the second map (figure 5.4) to be more meaningful, but considered the categories to be too general and asked the system to use more specific terms.

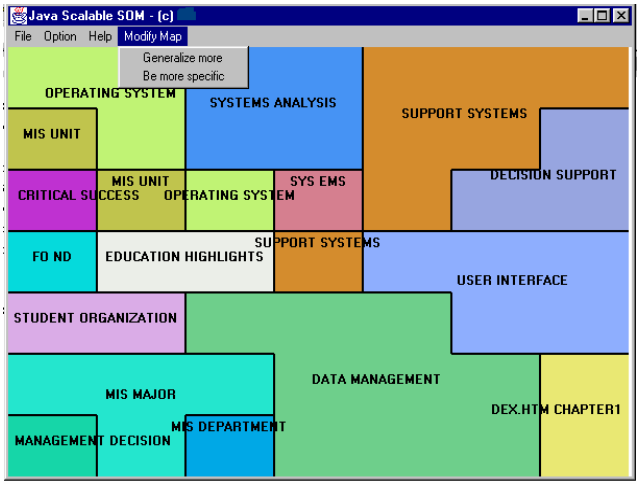


Figure 5.5. The concept map for the query "MIS grants" after user requested more specific terms.

The resulting map is shown in figure 5.5. This new map had 12 categories. The user agreed that at this point she could grasp a high-level picture of what concepts were

present in the collection of pages presented by search engine. She requested documents belonging to the categories “Operating Systems,” “Decision Support,” and “User Interfaces.” The user confirmed that the documents mapped into those regions indeed contained issues related to the corresponding concepts.

5.6.1.2 “Chloroplatinic Acid Density” Task

In the second example, the user (a chemical engineer) was looking for some very specific information about *chloroplatinic acid density.* She was not confident about correctness of her terminology. The user tried the Alta-Vista search engine first with a query "chloroplatinic acid density.” The summaries of the top 20 documents did not seem promising so she gave up browsing the search results and switched to CSOM. The same query produced the categories: CHEMICAL, DENSITY, ACID, DATA. She decided to remove all of those terms since they did not give additional information about the search results, although seemed relevant. The rebuilt map had concepts: `MATERIAL, EFFECT, LIQUID, NBSP, PRODUCT, APPLICATION, VOLUME, PRODUCT, TABLE, CATEGORY.` The user noted, that those terms describe the search results better. All of them seemed relevant except `NBSP,` which was just a result of an incorrect automatic indexing of HTML documents (an artifact). These relevant terms triggered query refinement. The user entered another query: "Chloroplatinic acid density concentration-tables.” She explained that "density concentration-tables" corresponded better to her information need. The resulting map consisted of concepts `TABLE, DENSITY, and DATA.`

Based on the same reasoning as for the first map, she removed all of them and obtained a map with about 20 concepts. One of them (`CONCENTRATION-TABLES`) captured her attention. She browsed all 6 documents belonging to this concept. Only one of those documents was very close to her information need. It referred to a website that was very helpful for her information forage. The user decided to explore that particular site and to stop using our interactive search system at that point.

5.6.2 Log Files Analysis

In this section we report our analysis of usage patterns that we obtained from the log files. Since our demonstration prototype was implemented as an Internet CGI application, most user actions were automatically recorded by our HTTP server that usually stores the URLs of all requested pages along with host computers that requested those pages. Our trace file used for this paper is available at: http://ai.bpa.arizona.edu/resume/dmitri/traces.txt

It contains the data collected over the period from April to September 1999.

Number of sessions analyzed	50
Total number of queries	113
Total number of requests for adaptive features	118
Proportion of sessions where users executed the adaptive features	66%
Proportion of sessions with query refinements	18%
Proportion of sessions with query refinements performed after the adaptive features have been used	16%

Table 5.2. The statistical summary of the analysis of 50 search sessions.

Table 5.2 presents a summary of statistics gathered from the recorded search sessions, which is discussed in detail below. Judged by IP addresses of the users, we estimated total number of unique users to be between 30 and 40, less than number of sessions since some users returned to the demo multiple times. The majority of our users have been web surfers who browsed our web site and encountered the Interactive web search demo. Based on the data stored in the log files we have been able to closely investigate the following questions:

1. Do category maps in general communicate high-level information about search results?
2. Are adaptive features used at all?
3. What adaptive features have been used most?
4. What terms do users tend to remove from category maps?
5. What proportion of interactive sessions seems to be successful?
6. Do category maps help in query refinement?
7. Do adaptive features add something to the query refinement process?

We defined each session as a sequence of requests from the same host computer. Each session started with a query. If a session contained several queries, all queries must have shared at least one word. If they did not, we assumed that the user changed the topic of search and thus started a new search session. We provide our analysis below.

5.6.2.1 Do category maps in general communicate high-level information about search results?

As we wrote in section 5.3, while browsing Internet search results, most users go no further than the first 10-30 documents before quitting in order to reformulate a query or simply give up. Since a typical response from a web search engine contains thousands of documents, more than 99% of the response to the query is left unexplored.

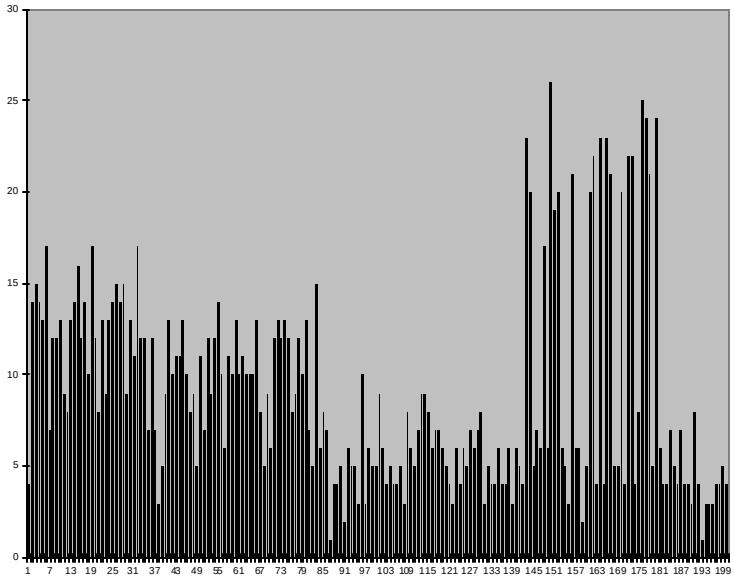


Figure 5.6. The histogram of positions of documents requested by users from the map in the ranked ordered list presented by search engines.

Figure 5.6 shows a histogram of positions of documents requested by users from the map in the ranked ordered list presented by search engines. Based on the category maps, users requested not only the first 10-30 documents, but all 200 documents identified by CSOM with almost uniform probability. This suggests that:

1) Documents that seemed relevant to users are not always in the top 10-30 returned by a search engine, and thus would remain unnoticed in the absence of visualization.

2) Category maps promote access to documents even beyond the top 10-30.

5.6.2.2 Are adaptive features used at all?

In 33 out of 50 sessions analyzed, users employed at least one of the adaptive features mentioned in the preceding sections. For 113 queries performed in all sessions, users requested adaptive features 118 times, 2.36 times per session on average.

5.6.2.3 What adaptive features have been used most?

We counted total number of requests for each feature. If a feature was used multiple times inside the same session it was counted multiple times. In the order of frequency of use, the features rank the following way:

"Remove terms"	50
"Please be more specific"	37
"More concepts"	15
"Please try to generalize more"	13
"Fewer concepts"	3

5.6.2.4 What terms do users tend to remove from category maps?

We analyzed a list of terms users asked to be removed and observed the following categories:

1. **Totally irrelevant terms (40%).** For example, from the map built for a query '`IBM disk-drive`[1]" the user removed the terms *ancient civilizations,* and *interface*. These terms seem to have been the result of the large amount of noise present in Web documents. Sometimes AI techniques fail to remove noisy terms from category maps automatically. Asking the user to remove them manually serves as a last resort.

2. **Relevant but not helpful terms (20%).** Terms that seem to be relevant to the topic of the search but that (as the user has decided) do not help in seeing a high-level picture about the search result fall into this category. For example, a user browsing the map created for the query "israeli palestinian conflict" removed the term *abu marzook* in-spite of this term's being the name of a Hamas officials mentioned many times in the context of 'Israel-Palestinian." In many cases, we can attribute removal of certain terms to a user's non-familiarity with their meanings.

3. **Trivial terms (25%).** Such terms seem to be relevant to the search topic but do not add anything to the description of the search results. For example, terms *grant*, and *management* were removed from the map built for the query "MIS grants." Term *IBM*

[1] We present queries in the Alta-Vista "simple query" syntax. So, *disk-drive* means a phrase "disk drive."

was removed from the map built for the query "IBM disk-drive." Term *mouse* was removed from the query "mouse." The presence of those terms on the map did not give users any new information since users already had deemed them relevant to the topic *a priori*. In 90% of cases, such words enter user queries. So, automatically discarding terms used in the user queries would be a good improvement to techniques based on category maps.

4. **Artifacts (15%).** Such terms seem to appear solely due to lack of precision in the automatic indexing procedure. They could be removed based on more accurate conversion of web documents (written in HTML language) into text sentences. Examples of such terms from our traces: *coverage:world news:israeli, nbsp, thread.*

5.6.2.5 What proportion of interaction sessions seems to be successful?

Although in this research we did not intend to measure the success rate of search sessions, as indirect evidence we would like to suggest that the proportion of sessions in which users requested documents serve as a proxy. From previous research, users who dislike search result summaries tend to abandon the query rather than investigate the full-text results. From our bg files we found that users requested documents in 38% of the sessions.

5.6.2.6 Do category maps help in query refinement?

We can answer this question by counting the number of sessions in which users changed their queries after looking at category maps. We counted nine such sessions, which constituted 18%. Total number of refinements registered in our log files was 63.

5.6.2.7 Do adaptive features add something to query refinement process?

In 8 out of 9 sessions involving query refinement, the refinement was done after adaptive features had been used. This indirectly indicates that adaptive features help to make refinement decisions.

5.7 Observations, Conclusions and Lessons Learned

It has been established in prior research that automatically created through document and term clustering, maps of concepts improve navigation in large collections of text. The documents found through this navigation do not, however, always match users' expectations. We have suggested leveraging this approach by providing the user's ability to interactively customize a maps, thus overcoming some of the quality problems associated with unsupervised self-organizing maps.

For this purpose we have created and tested a prototype Customizable Self-Organizing Map (CSOM) that builds and refines in real-time a map of concepts found in Web documents returned by a commercial Internet search engine. We used AltaVista (www.altavista.com) but the approach may work with other search engines as well. By

observing two search sessions and analyzing trace files from 50 search sessions we found that users *employed and had success with customization features*.

Since we had shown how the problems associated with quality of applications of automated document clustering can be alleviated, we believed that we have increased the chances of overall success of an interactive information access system based on clustering and concept maps.

We concluded that our prototype could be used for the empirical study described in the next chapter after several modifications, which were inspired by our observations and lessons learned. In particular:

1. The users complained that they had difficulty navigating the maps region by region. If the desired web page was not in the explored region, proceeding with navigation required backing up and selecting another region. This finding is in line with prior research reported by (Chen et al., 1996; Hearst, 1999). In order to overcome this, we proposed using the map to elicit user feedback and create a list of documents ordered by relevance based on this feedback.

2. In spite of users' understanding of a graphical interface, many of them suggested that a simpler, HTML based implementation like that Internet search engine use would be more intuitive. For this and other reasons explained in the next chapter, we implemented our next prototype based on a simple and well known HTML form interface.

3. We observed and the users noticed that they had difficulty composing initial queries. For our next study, we decided to give the users the ability to start with a simple textual description of their information need. No knowledge of keywords or Boolean syntax would be necessary in that case.

We also came to the conclusion that hosting the visualization component and the search engine, as we did in using AltaVista search engine (www.altavista.com) on different servers, requires extensive data exchange and thus with current bandwidth is not fast enough for an interactive application. The future solution to this problem may be to host the visualization layer at the same server where search engine is. Commercial Internet search service providers such as Yahoo or AltaVista would not have problems implementing it this way. Our solution for the purposes of studying was to pre-build the maps for the queries used in our experiment.

The next chapter presents our empirical study, concluding the overall paradigm evaluation.

6. SOM AND INTERNET SEARCH

6.1 Objectives

As we noted in the "Literature Review and Research Formulation" chapter, summarization and clustering tools have been shown to be helpful in visualizing large volumes of textual data. However, no experimental evidence has yet shown that this approach leads to faster or more effective searching.

In our previous chapters, we explained the rationale behind the approach that we have been researching and how we have resolved many issues related to computational speed and quality of output from the techniques that we have been using. This chapter plays a key role in the dissertation. It addresses our central research question concerning *whether automated clustering of text documents can help interactive information seeking* by empirically evaluating Adaptive Search, the interactive approach that we have developed, to cluster and summarize data and incorporate user feedback.

Adaptive Search uses a Kohonen's self-organizing map (Kohonen, 1995) as a clustering and summarization tool and interacts with the user via HTML forms and ordered lists of documents. The forms provide cues to the user about the pages found by a commercial search engine and accepts user feedback in order to direct the search.

Our study compared search performance when Adaptive Search was used to augment a commercial search engine by means of an intermediate visualization layer against direct

use of a commercial search engine. In a laboratory experiment, subjects searched the World Wide Web for answers to a set of given questions. We found that subjects required less time to reach correct answers using Adaptive Search than using a commercial search engine directly. We also found that Adaptive Search consistently positioned documents containing answers higher in rank-ordered lists than the commercial search engine did. Our findings suggest that document clustering, summarization, and feedback can offer significant benefits when integrated into an interactive search system.

6.2 Research Questions and Methodology

The study reported in this chapter answers the following research question: "*Does Adaptive Search help information seekers*?" We compared using Adaptive Search to augment a commercial search engine against direct use of the same commercial search engine. Since the latter follows the query reformulation steps discussed on page 28, we call the approach using the commercial search engine the "query based search" (QBS), to distinguish it from our approach. We called our approach Adaptive Search (AS) because we believe that the system adapts to the user and the user's search task through interaction.

We performed a laboratory experiment in which subjects searched the entire World Wide Web for answers to the questions discussed by the panel on Web Search at the 1998 ACM Conference on Advances in Information Retrieval (listed later in this section in Table 6.3). Tasks were very specific and clearly defined, mostly related to entertainment/tourism topics. For example: "What does it cost to ride on the upper deck

of the Star Ferry across Hong Kong harbor to Tsimshatsui?" The tasks represented the "needle-in-haystack" situation, in which only 2-3 pages out of millions of sources on the web might contain the answer to the questions. Since the tasks were very unambiguous it was easy for the supervisor to decide whether the answer had been found.

We intended to find out both how much our technique helped (quantitative scale) and why it helped (qualitative scale). For quantitative analysis we developed numerical metrics based on prior research and followed the traditional hypothesis testing methodology. We called a set of measurements obtained from the search sessions involving Adaptive Search (AS) a *treatment group*, and those obtained from the sessions involving Query Based Search – a *control group*. In addition to numerical metrics, we also asked for users' preferences between the two tools. For qualitative analysis we processed the usage patterns extracted from our trace files in order to find examples supporting our assumptions underlying the effectiveness of our approach (page 123).

6.3 Testbed

We have already explained why we chose the Internet as our test-bed. It is worth adding that, although we did not find answers to the search tasks in any of pages listed in the Yahoo Internet directory (www.yahoo.com), we believe that the answers were not there -- at least at the time of the experiment -- because the questions were very narrow ("needle in a haystack"). This observation justifies using AltaVista (www.altavista.com) or another search engine with a larger database of indexed pages than manually crafted Internet directories.

6.4 Prototype Design

6.4.1 Rationale Behind

We deliberately designed and implemented a prototype for the empirical study described in this chapter, which we believe can also be used for real-life web searching after some modifications, provided it is hosted on the same server as the underlying search engine. Otherwise, a very fast network connecting link has to be provided. For the purpose of the experiment, we pre-built the maps for the search tasks that we used.

We purposely simplified the interface to overcome distraction by the visual appeal of the 2D or 3D graphical interface and to respond to users' suggestions described in the preceding chapter. Since today's Web search engines provide an HTML form interface, we sought to ensure a fairer comparison.

Our "Literature Review and Research Formulation" chapter (page 40) explains why we believe that Adaptive Search has potential to be more effective than previously explored tools. Here, we would like to summarize that for our approach to be effective the following should hold:

1. The system is able to describe clusters using terms extracted from the current search results.

2. Users are able to distinguish quickly terms describing relevant and irrelevant documents, and correctly mark those terms.

3. The system is able to deliver an efficient summary of current search results to the users, so they can notice what is missing for their information need.

4. The users are able to describe the missing information.

Below, we describe how we achieved this.

6.4.2 Commercial Internet Search Engine Features

Before explaining how the system (Adaptive Search) interacts with the user, it is appropriate to describe the features of the commercial search engine it used (AltaVista).

Most commercial Internet search engines accept queries in the form of text strings, composed according to rules (syntax) specific to each engine. The engine makes a guess about the user's interests and returns a list of documents, ordered by the perceived level of interest. The list usually includes titles, URLs, and 3-4 sentence summaries of found web pages called *snippets*. An example of such a list returned by AltaVista is shown in Figure 6.1. A rather comprehensive and regularly updated review of most popular engines can be found at the web site www.searchenginewatch.com.

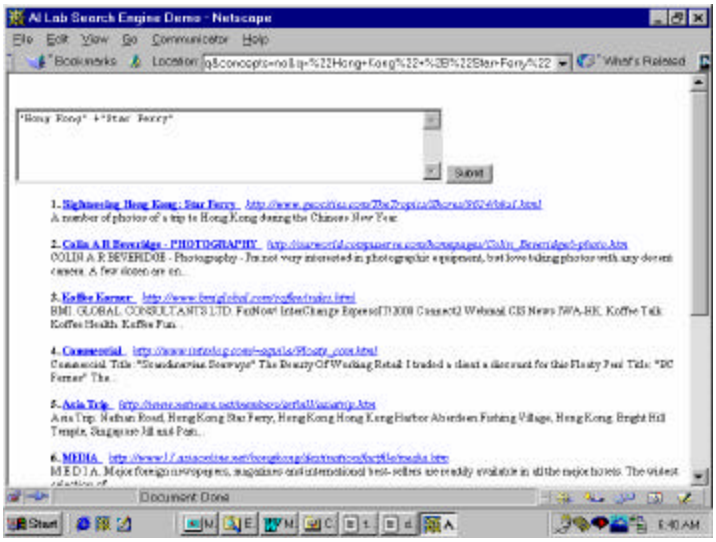


Figure 6.1. A list of documents returned by a query based search.

Since the underlying matching algorithms are proprietary and frequently changing, general public does not usually know them. We list below the features that we used in our study. Most search engines, including "Simple Search" AltaVista interface (http://www.altavista.com/), support those features.

Feature 1. *Ability to specify what words or phrases should influence rank order*. Users of AltaVista "Simple Search" do this simply by entering those words or phrases in the query. For example, the query "hong kong tsimshatsui" results in the placing of documents containing the words "hong," "kong," and "tsimshatsui" closer to the beginning in the rank ordered list of found pages. The more occurrences of those words in a page, the higher chance for the page to be at the beginning. Search engines also use metadata to influence the rank order. For example, the presence of those words in its title would rank a document higher.

Feature 2. *Ability to require certain words or phrases to be in the matching pages.* In the AltaVista "Simple Search," this is done by placing the "+" sign right before the word or phrase. For example "+hong +kong tsimshatsui" would find only documents containing both "hong" and "kong." This is equivalent to the "AND" operator in many boolean query languages: "hong AND kong." In "Simple Search," words preceded by the "+" sign also affect rank order, as explained in the preceding paragraph. So, the engine not only returns pages containing all of those words, but also it orders them such that pages containing many of those words would appear at the beginning of the list.

Feature 3. *Ability to exclude pages containing certain words.* In AltaVista "Simple Search," this is done by placing the "-" sign right before a word or phrase in the query. For example, "+hong +kong tsimshatsui -view" would never return any pages containing the word "view."

Feature 4. *Ability to return number of indexed web pages containing the specified words or phrases.* This feature is intended to help users refine their queries. This number's being very large indicates that the word is too general and probably not very useful in queries. This number's being very small or zero usually indicates a spelling error or other reason for the word's being unpopular on the web.

6.4.3 Interaction Between The User And The System

This section describes the Adaptive Search approach. Interactions among the user, the system, and the search engine are shown on Figure 6.2 and explained below. Section 6.5.5 presents an example of a search session.

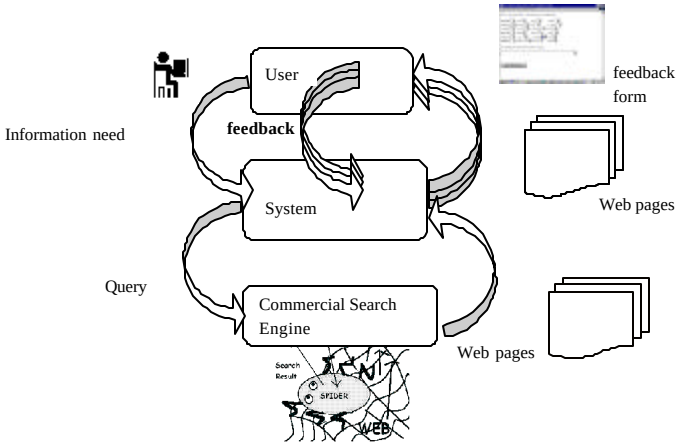


Figure 6.2. Adaptive Search approach.

Step 1. The user submits to the system a simple text description of the information need. No query language is required. The system sends the same description to a commercial search engine (SE), thus relying on Feature 1 of the search engine. The search engine returns a list in ranked order of documents matching the query.

Step 2. The system fetches (from the Web) the 200 top documents in the ordered list and builds a self-organizing map for them. The self-organizing map algorithm automatically clusters documents and assigns labels to the clusters. Based on the information contained in the map, the system generates a summary, presented to the user in HTML form. The next section provides more details about this form.

Step 3. The user marks labels and terms on the summary form according to their relevance to the current information need, and may type additional words or phrases describing the information believed to be missing in the summary. The next section provides more details.

Step 4. Based on the selected and entered terms the system creates a set of queries and sends them to the search engine. Section 6.4.5 describes how the system does it. Then, it presents a rank-ordered list of documents to the user.

Step 5. The user browses the rank ordered list of documents and inspects the pages that seem promising based on the snippets that the commercial search engine provides.

Step 6. Iterate: The user may fill the feedback form differently and re-submit it, thus going through steps 3-5 repeatedly in order to find the pages of interest.

6.4.4 Feedback Form

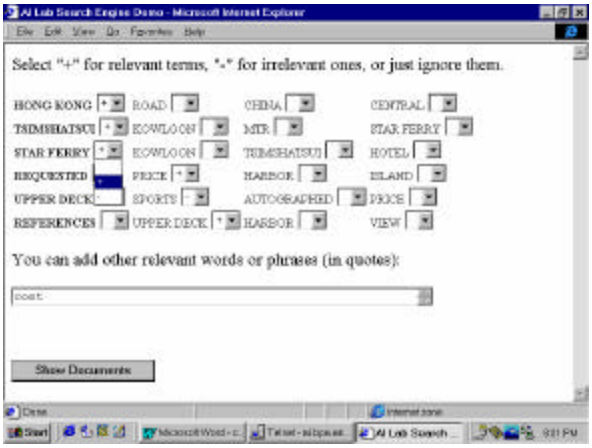


Figure 6.3. An example of an HTML form generated by Adaptive Search for the Star Ferry task: "What does it cost to ride on the upper deck of the Star Ferry across Hong Kong harbor to Tsimshatsui?" The user marked the terms `hong kong, tsimshatsui, upper deck, price, star ferry` as close to and the term `sports` as "far from" the information need. The user also added the term `cost`, to additionally direct the search.

Figure 6.3 shows an example of an HTML form used in Adaptive Search. One line of the form describes one region (cluster) in the self-organizing map. The first word or phrase (in boldface) labels the region, created by the algorithm as described in the preceding section. The next three most representative terms for the region follow the label term. In this implementation, we had discarded other valuable properties of self-organizing maps, such as visual representation of proximity between topics (clusters). We used only clustering and key concept extraction capabilities. Other clustering and summarization tools can be used as well. An example is the earlier mentioned Scatter/Gather (Cutting et al., 1992).

Users provide feedback by marking words or phrases as “close to” or “far from” the information need, using the “+” or “-“ sign from a pull-down menu.

The descriptive term list (6 x 4 = 24 in the example in figure 6.3) double-functions as a document summary: The user can add any missing key topics on the bottom line (Figure 6.3).

6.4.5 The User Feedback Component

This section explains how the system (Adaptive Search) uses the feedback provided by the user. After a number of preliminary studies, we eventually settled on a heuristic algorithm that gave the best results. The general idea behind the algorithm is to first create a so-called “ideal query” by combining all user feedback. If the “ideal query” returned too few documents (10 or less for the first cut in our current implementation) the algorithm modified this “ideal query” in order to get enough documents.

The algorithm constructed the “ideal query” such that the matching documents:

1. Would have all the words the user marked as “close to the information need” or entered on the additional line. To achieve this, the algorithm used Feature 2 described in the preceding section.

2. Would not have any of the words the user marked as “far from information need.“ To achieve this, the algorithm used Feature 3 described in the preceding section.

3. Would be rank ordered according to the words marked as “close to the information need,” words entered on the additional line or entered at the very beginning (step 1 on page 127). To achieve this, the algorithm used Feature 1 described in the preceding section.

The following example helps to clarify this algorithm. At the very beginning (step 1 on page 127) the user typed the sentence “what does it cost to ride on the upper deck of the Star Ferry across Hong Kong harbor to Tsimshatsui?” Adaptive Search created and displayed the HTML form shown in Figure 6.3. The user marked `hong kong`, `tsimshatsui`, `upper deck`, `price`, and `star ferry` as “close to the information need.” The user also marked the word `sports` as “far from information need,“ and typed the word `cost` in the additional input line. The “ideal query” expressed in AltaVista syntax was:

```
What does it cost to ride on the upper deck of the Star Ferry across Hong Kong harbor to Tsimshatsui +“hong kong” +tsimshatsui +“upper deck” +price +“star ferry” +cost –sports
```

According to the search engine syntax, this requires matching documents to have the words “hong kong,” “tsimshatsui,” “upper deck,” “price,” and “star ferry.” The matching documents will not have the word “sports.” Also, the presence of any of the words “what” “does” “it” “cost” “to” “ride” “on” “the” “upper” “deck” “of” “star” “ferry”

"across" "hong" "kong" "harbor" "hong kong" "tsimshatsui" "upper deck" "price" "star ferry" "cost" would make the rank of matching documents higher.

To obtain more matching documents, the algorithm modifies the "ideal query" by removing some of the required (prefixed with "+" in the ideal query) or excluded (prefixed with "-") words/phrases. The algorithm first removes the words and phrases found most frequently on the Web. The frequencies of occurrence of a word or a phrase in entire web (called *document frequencies*) are determined by querying the search engine (Feature 4 in the section 6.4.2). Table 6.1 shows the document frequencies for the required terms from the example above.

Term	Number Of Pages Containing It
hong kong	1,534,031
tsimshatsui	2,578
upper deck	69,226
price	18,241,027
star ferry	593
sports	10,428,227

Table 6.1. Document frequencies on the Web for terms related to the Star Ferry task.

The table indicates that the phrase "star ferry" is very rare, while the word "price" is very common. So the algorithm would drop the term "price" before dropping "star ferry."

If several terms had to be dropped, the algorithm followed a heuristic strategy based on the widely used "inverse document frequency" weighting (Salton, 1983). The strategy of our algorithm was to maximize the "objective function," which was the sum of $log(N/df_i)$ for each term i remaining in the query. Here, df_i is the number of web pages containing

the term i (document frequency) and N is the maximum df_i among all the terms in the "ideal" query: $N = max\{df_i\}$, for $i = 1$ to M, where M is the number of required words and phrases in the "ideal query." This ensures that the logarithm above always exists.

This strategy would discard the most frequently occurring words, since they would have smaller weights. We understand that there can be many other possible implementations incorporating user feedback into the ranking algorithm. The current heuristic implementation is a result of our preliminary empirical studies and observations of user strategies.

It should be noted that our algorithm does not perform any analysis of the initial task description. If it did, it would be much harder to argue that the overall effect established by the experiment is due to the visualization component and not to the advantage of having the initial search task description. We used it only to create a map, which served as a starting point, and to provide additional context for the queries sent to the search engine, as described above.

6.5 Experiment Design

6.5.1 Assumptions

Thirty-six (36) undergraduate students in the school of business agreed to be subjects in this study. They represented users with a wide range of computer skills, from novices to Web search experts. Their skill and experience with commercial search engines was pre-tested by the questionnaire. Eight subjects received $5 for participation, 10 subjects

received extra credit for a class, and the rest volunteered. Since the reward did not depend on subject performance and the tasks were randomly assigned, we believe that we can safely ignore those differences in conditions, considering that subject's ability was a random factor anyway. Once a subject agreed to participate, for whatever reason, the productivity incentives for the subjects were the desire to finish earlier and the feeling of self-satisfaction with the accomplished task.

We assumed search performance to be a function of the tool (QBS or AS), a particular subject, and a particular search task. Each subject was asked to find the answer to two of the total set of 10 search tasks used in the study. The tasks were randomly assigned. Order of tasks and interfaces was properly balanced to counter carry-over learning effects.

6.5.2 Procedure

Since the implementation of a truly multi-user system would incur additional implementation costs we decided to ask one subject to participate at a time.

Along with "real" search tasks for which the subjects were expected to find a web document containing an answer to the search question, each subject also performed eight "virtual" search tasks (the remaining tasks out of 10 tasks used in the study) that the subjects started but were stopped after 1-2 minutes. Virtual tasks were less time-consuming and thus more efficient in providing data. The "Metrics" section explains how these data have been used.

Time for each search task was limited to 15 minutes. The experimental procedure with each subject was the following:

1. Fill out pre-questionnaire. (5 minutes)

2. Perform tutorial. (10 minutes)

3. Perform 1st task with interface 1 (QBS or AS). (up to 15 minutes)

4. Perform second task with interface 2 (AS or QBS). (up to 15 minutes)

5. Perform eight virtual tasks while switching interfaces. (approximately 15 minutes)

6. Fill out post-questionnaire. (5 minutes)

The textual description of a search task served as the starting point for both interfaces. The supervisor gave the task description to the subject. If the subject used Adaptive Search, he/she entered the description into the system (step 1 on page 127), received the HTML form from the system (step 2 on page 127) and provided feedback by marking words or phrases as “close to” or “far from” the information need. Then the subject submitted the form and received an ordered list of documents, which he/she explored to find the answer. The subjects were allowed to do repeated form submissions (step 6 on page 127).

A subject using Query Based Search was free to enter a query using AltaVista “Simple Search” syntax, which allows entering the unaltered text description of the task. Entering

entire text description never resulted in finding the answers. Very few subjects actually followed that strategy.

The tutorial consisted of explaining Alta-Vista's simple query syntax and search strategy. The supervisor explained the notion of rank order, use of the "back button" in the web browser, and the "find inside a page" functionality. Each subject was asked to find the answer to the question "What is the capital of Honduras." Then, the Adaptive Search approach was explained, using the same tutorial task. The supervisor spent approximately the same amount of time for the tutorial using each interface and followed the same script with each subject.

All user actions such as buttons pressed, queries entered and pull-down menu selected were automatically recorded by the server's CGI script, along with all the web pages visited by the subjects. The supervisor recorded timing.

6.5.3 Metrics

Prior research has produced many useful metrics to evaluate interactive information access systems beyond the one traditionally used in Information Science, precision and recall (Salton, 1983). The alternative metrics include time required to learn the system, time required to achieve goals on benchmark tasks, and error rates (Hearst, 1999).

We measured the time it took to find the answer to a given question. We assumed that a better system requires less time for information seeking.

We analyzed our log files after pilot experiments and came to the conclusion that the Adaptive Search system itself spent considerably more time processing, 95% of it for multiple interaction sessions with the commercial search engine (AltaVista). This time expenditure would not be necessary if Adaptive Search were located on the same server as the search engine. Also, we believe, for two reasons, that more accurate metrics could be based on the time that the user spends searching, ignoring waiting time. In the first place, waiting time is a function of underlying technology and may be easily reduced, and second, waiting time does not impose the same cognitive load as active interaction does: The user may spend that time on other activities.

This is why we selected user searching time, which we obtained from our log files, to be our primary metric. We limited each session to 13 minutes (user time). Although we allowed some subjects to spend more time searching, we reduced to 13 minutes any time longer than that for analysis purposes. We considered the task accomplished within the time allowed only if it was accomplished within 13 minutes (user time).

To evaluate the quality of the returned rank ordered lists of documents, we used a different metric. We analyzed rank ordered lists returned by both systems when subjects performed "virtual" tasks to find the first page containing the answer to the question contained in the task. We called the position of this page in the list *Answer Rank*. Ideally, Answer Rank should be 1. This metric is less direct than the one based on time, but is more stable since it depends on fewer random factors in the experiment, such as web traffic and subject's ability to comprehend documents.

We chose this metric rather than the traditional Information Science notion of *precision* (Salton, 1983) because of our main purpose. We were interested in having the subject find at least one document containing the answer to the search task, but not in composing "efficient" queries that matched as many answers as possible. Precision and recall measures have been widely used for comparing the ranking results of non-interactive systems, but are less appropriate for assessing interactive systems (Lagergren & Over, 1998). In addition, further problems arise with very large collections in which the volume is such that no individual or group of experts could possibly identify all the relevant information.

We also analyzed the number of pages that subjects visited in order to find the answers, including the pages with search results returned by the systems (AS and QBS). We analyzed only the cases for which subjects found answers.

Since we limited the time that subjects could spend searching, we believed that another useful metric would be the proportion of tasks that were accomplished in the given time.

6.5.4 Hypothesis

Our **null hypotheses** are listed below:

H1: It takes the same user time to do the tasks with either tool.

H2: It takes the same physical time to do the tasks with either tool.

H3: It takes the same number of pages to visit to find the answer if the answer has been found in the time allowed.

H4: Using both tools results in the same AnswerRank.

The alternative hypothesis were that AS was better as measured by the metrics listed above.

6.5.5 Example

This subsection presents an example of a search session. The subject typed in the question “What does it cost to ride on the upper deck of the Star Ferry across Hong Kong harbor to Tsimshatsui?” and received the form shown in Figure 6.3.

As it turned out, “Upper Deck” is also the name of a company that makes sports equipment, which explains why some sport-related concepts appeared in the summary. The subject marked with “+” the concepts: “hong kong,” “tsimshatsui,” “upper deck,” “price,” “star ferry,” and with the “-“ the concept “sport.” After pressing the “Show Documents” button, the subject received the list of ranked documents shown in Figure 6.4. The second web page contains the passage shown in Figure 6.5, which clearly contains the answer to the search question.

Figure 6.4. A list of documents found by Adaptive Search for the Star Ferry question.

Figure 6.5. A passage fron containing the answer to tl question.

Another subject was asked to use a query based search form and entered:

“hong kong” +”Star Ferry”

This query resulted in the list of documents already shown on Figure 6.1. The documents were mostly about sightseeing from the Star Ferry and did not mention either the cost of the ticket on the upper deck or Tsimshatsui as the destination. An example of a more efficient query would be:

+"hong kong" +"tsimshatsui" +"star ferry" +"upper deck” cost

Very few subjects were able to compose a query resembling this one. The potential pitfalls already have been mentioned in the "Literature Review and Research Formulation" chapter: forgetting keywords, mismatched vocabularies, wrong use of syntax, too short or too long queries that result in too many or no documents.

6.6 Results and Discussion

Table 6.2 display several overall statistical results that are discussed below in this section.

metric average	Query Based Search	Adaptive Search	hypothesis	t-test, p-value
user time	8.7 minutes	7.3 minutes	reject H1	0.08
physical time	13.1 minutes	11.9 minutes	can not reject H2	0.19
number of pages visited	5.07	4.17	reject H3	0.090
answer rank	12.86	10.60	reject H4	0.00224
proportion of tasks accomplished	15/42	21/42		

Table 6.2. Overall statistical results: Visual Search vs. Query Based Search.

6.6.1 Searching Time

While analyzing time spent searching, we decided to ignore tasks 6, 8, and 10 since no subjects had been able to find the answers to them within the time allowed. For the remaining tasks, the average time subjects spent using Adaptive Search was 7.3 minutes. The average time spent using Query Based Search was 8.7 minutes. A t-test of the difference between means resulted in p-value = 0.08, a significant difference at alpha = 0.1.

Adaptive Search was faster (11.9 vs. 13.1 minutes) even when “raw” (physical) time was considered, but the difference was not statistically significant, resulting in p-value = 0.19.

Those two results provide evidence that Adaptive Search requires less time, despite high variation due to a large number of random factors involved, such as differences in web transmission times, subject skills, and task difficulties.

We observed that the subjects spent approximately 98% of their time reading web documents or search result pages, and only approximately 2% on typing or making menu selections. Thus, different amount of typing while working with different interfaces is extremely unlikely to explain the above result. We believe that the difference in average time spent searching has been due to different paradigms: traditional Query Based search against Adaptive Search.

Task #	Task Description	Average Answer Rank		t-test, p-value	Adaptive Search is	
		Query Based	Adaptive		better	worse
1	I want to find where Max Beerbohm, the English caricaturist, lived in at the end of his life.	12.38	13.29	0.34		
2	What does it cost to ride on the upper deck of the Star Ferry across Hong Kong harbor to Tsimshatsui?	10.69	2.91	3.97E-04	√	
3	Where can I get a good pfeffersteak in Hagerstown, MD USA?	13.21	6.94	1.31E-02	√	
4	If I visit Singapore, what, if any, buildings designed by I. M. Pei's can I see there?	6.16	11.56	7.71E-03		√
5	Names of hotels in Kyoto (Japan) that are near the train station?	15.81	15.29	0.39		
6	What is the cost of overnight train tickets, including sleeper accommodations (double occupancy) from Paris to Munich?	20	20	0.5		
7	How long does it take to get by train from Copenhagen to Oslo?	10.73	9.88	0.36		
8	Was the Ring Cycle performed at Bayreuth, Germany, in summer 1998?	20	20	0.5		
9	I'm looking for the names of campgrounds around Lake Louise (Canada) that have showers.	16.94	10.61	7.76E-04	√	
10	I need a map showing the location of the Penfold's winery in Australia.	19.31	14.35	1.37E-02	√	

Table 6.3. Average answer rank for each search task.

6.6.2 Search Results Quality (Answer Rank)

We had decided to check only the top 19 pages to see if they contained answers to the given questions. We assigned an Answer Rank (AR) of 20 to those lists that did not contain the answers in any of the top 19 pages. As a result, in order to obtain Answer Ranks, we analyzed about 40*10*20 = 8000 pages. This analysis required a total of approximately 80 person-hours. A person responsible for analyzing pages did not know from which system each particular page had come, so there was no bias toward either of the systems.

Overall, Answer Rank was better for Adaptive Search: 10.60 vs. 12.86. The t-test for the difference resulted in p-value = 0.002. Table 6.3 shows average Answer Rank for all 10 tasks. Answer Rank was significantly better for Adaptive Search in tasks 2, 3, 9, worse in task 4 and not statistically different in other tasks. Answer Rank was the same and equal to 20 for tasks 6 and 8, since no subjects were able to obtain answers for those tasks in the top 19 pages with either interface.

We also ran similar tests using reciprocal Answer Rank, in which the results should be much less sensitive to the cut-off (19 pages in our case) because the tails have less influence on the sample mean. For example, the difference between 1/20 and 1/30 is much smaller in absolute terms than the difference between 1/2 and 1/3, so the exact cut-off number is much less important. We obtained the same result for the Reciprocal Answer Rank (RAR), with t-test p-value = 0.011.

To address concern about using repeated measures (collected from the same subject in different tasks), we also analyzed data arranged subject by subject. We computed the "effect" for each subject as the difference in average reciprocal answer ranks:

effect = average RAR using AS - average RAR using QBS.

It is easy to see that those effects were independent. They are displayed in the Figure 6.6. The mean effect (0.12) was positive (t-test p-value = 0.013).

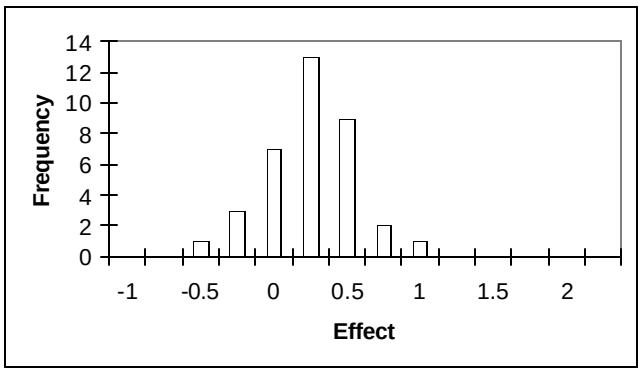


Figure 6.6 Histogram of the effect by subjects.

Notably, the average answer rank achieved by the subjects using QBS after their second query submission was also inferior to the one achieved by the very first submission of feedback by subjects using AS: 14.5 vs. 10.2 (only "real" sessions considered), t-test p-value = 0.0026 Thus, even after looking at the rank ordered list and sometimes several documents at the top of that list, the subjects were not able to modify their queries to achieve the same relevance of query results as was demonstrated by a single submission of the HTML form by AS users.

We also tested whether outcome was sensitive to search task. Even when we removed task 2, which resulted in the best AR for Adaptive Search, we still obtained better results for Adaptive Search than for QBS (p-value = 0.026).

These results indicate that Adaptive Search consistently positions the user closer to finding an answer than does Query Based search. This is another indication of the superiority of the new approach in this experiment.

6.6.3 Number of Web Pages Visited

We also analyzed the number of pages that subjects visited while searching for the answers, including the pages returned by both interfaces. We analyzed only the cases in which subjects found answers. The average number of page visits for the Query Based search was 5.07. For the Adaptive Search, it was 4.17. The t-test for the difference resulted in p-value = 0.090. This metric is more stable than time since it does not depend on web traffic and subject reading speed. It also indicates that Adaptive Search was revealed to be the more effective tool in the experiment.

6.6.4 Proportion of Tasks Accomplished

Out of 42 search tasks performed with each interface, 21 tasks resulted in finding the answer with the help of Adaptive Search, and only 15 tasks with Query Based search. This als*o shows that Adaptive Search was more effective.

6.6.5 User Preferences

Most of the subjects indicated in a questionnaire that they preferred Adaptive Search to Query-Based search. We asked them to choose an integer number between 1 and 5 to describe their degree of preference between the two tools, 1 corresponding to the strongest preference for QBS and 5 to the strongest preference for AS. The average subject preference turned out to be 3.6, which is statistically more significant than 3 (corresponding to indifference between those two tools) with p-value = 0.003. Subjects indicated that they preferred Adaptive Search over Query Based search because they did not need to come with keywords on their own, did not need to know query syntax, and had an HTML form with a set of concepts (created after entering textual description of the task) as a starting point for their search instead of the blank input line of QBS.

6.6.6 Obstacles to Better Effectiveness

Many subjects skipped pages containing answers after looking at the snippets (summaries) provided by the search engines or missed an answer in a web page containing it. These two facts may have contribute to lessening the time difference between the two interfaces, even when Adaptive Search consistently provided better lists of matching documents.

6.6.7 Easy vs. Tough Tasks

After pretests, we hypothesized that the effect would be stronger for more difficult tasks, defined as dealing with problems for which very few pages on the web contain answers.

We divided tasks into two groups of four, according to the number of answer-pages found through pretests. The experiment ignored tasks 6 and 8 since no subjects found answers to them.

For each subject, we computed the difference in effects between those measured by the "tough" tasks (1, 3, 9, 10) and by the "easy" tasks (2, 4, 5, 7). The mean difference was found to be statistically significantly more than 0 (t-test p-value = 0.013), which indicates that the effect of using AS over direct use of a search engine is stronger for the "tough" tasks. The difference is attributable to the worse QBS performance for the "tough" tasks (t-test p-value = 0.00043). The AS performance was not significantly different between the two task levels.

6.6.8 Why Adaptive Search Was Effective: Qualitative Analysis

Several examples confirm our theoretical assumptions underpinning the effectiveness of Adaptive Search (page 123). Below, we summarize them.

1. The system is able to describe clusters using terms extracted from current search results. For example, given the task "I'm looking for the names of campgrounds around Lake Louise (Canada) that have showers," the system used the terms "campgrounds," "trailer parks," "camping," "parks," "campground," and "facilities" for the summary of the search results, eliciting user opinion about relevance of those terms to the information need.

2. Users are able to quickly distinguish terms explaining relevant or irrelevant documents and correctly mark those terms. For example, the users correctly marked the terms mentioned in the preceding paragraph as “close to” the information need. Users also correctly marked words as “far from” information need, as in the case of the word “sports” for the Star Ferry question.

3. The system is able to deliver an efficient summary of current search results to the users, so they can notice what is missing for their information need. Users were able to describe missing information. For example, while working on the task “Where can I get a good pfeffersteak in Hagerstown, MD USA?” and being presented by the system with the summary shown on Figure 6.7, most users entered the word “pfeffersteak” on the additional input line, since the summary made no mention of pfeffersteak.

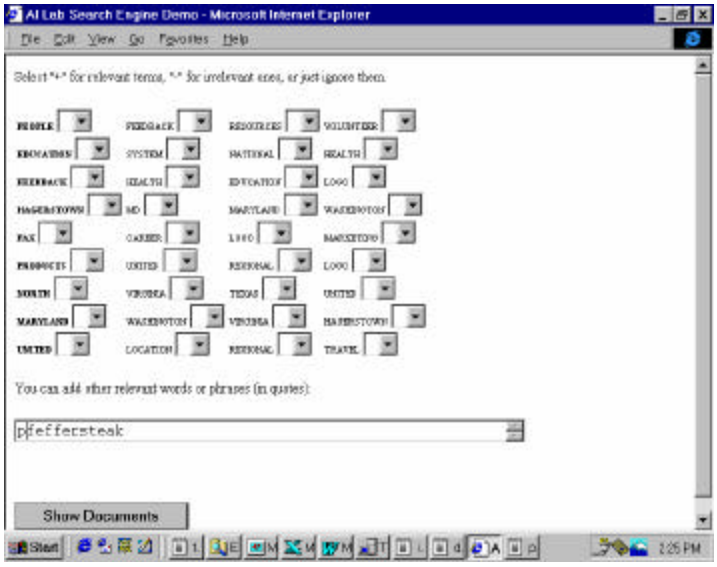


Figure 6.7. The feedback form for the task “Where can I get a good pfeffersteak in Hagerstown, MD USA?” Subject added the word “pfeffersteak” since it was not in the summary of clusters.

6.7 Conclusions

Based on the evidence described in the preceding sections, our approach to information seeking grounded on document clustering, summarization, and feedback seems promising, even superior to the traditional keyword approach. Our approach has many additional benefits: it does not require knowing a query language or being skilled in selecting appropriate keywords. The users found it intuitive, easy to use, and preferred using it to the keyword approach.

In our experiment subjects searched the entire World Wide Web using our approach (treatment group) and the search engine directly (control group). Since our current implementation acts as a layer between the user and a commercial keyword based search engine, we can conclude that the better search effectiveness revealed by subjects using Adaptive Search is attributable to this layer, which is based on clustering and summarization.

Due to the design of our experiment and our system architecture, we have been able to test only the overall effect quantitatively but did not differentiate separate contributions of several effects such as:

1. Set of clusters acts a summary, so users recognize what is missing in this summary and describe the missing information.

2. Summary of clusters described by terms elicits user's opinion on relevance of those terms to the information need.

3. Interactive search tool encourages entering textual description of the information need at the beginning of search process, thus the system acquires more context description from the user. The Alta-Vista, Internet search engine used by our control group, also allows but does not require starting with textual description.

4. Interactive search tool encourages more user input, thus it becomes easier for the system to come up with better results.

We argue that all the above effects are beneficial to the search system, adding value to the empirical result that our system aids information access through any of those effects.

Using the Web as a testbed entails some limitations. We have ignored that WWW is a hypertext repository, but treated each web page as independent document. Since the World Wide Web is a very large and diverse repository, we believe that our results are likely to be generalizable for smaller and more domain-oriented depositories such as enterprise intranets.

7. FINDINGS AND FUTURE RESEARCH DIRECTIONS

Using of information clustering to facilitate information access has been an objective aspired by researchers for a long time. However, evaluation results have turned out to be primarily inconclusive or negative. This dissertation reports a compilation of studies aimed at resolving many of the reported problems and developing a new interactive searching paradigm based on clustering as a summarization and feedback elicitation tool. We have been able to resolve problems related to the speed of the techniques used, the quality of their output and to confirm empirically that the proposed approach is superior to the traditional query based information seeking paradigm.

7.1 SOM Speed Issues

Our approach uses Kohonen's self-organizing map algorithm and acts as a layer between the user and a commercial keyword-based search engine. We have developed and tested a scalable implementation of a Kohonen's self-organizing map algorithm for the domain of free text analysis. Our proposed data structure and algorithm took advantage of the sparsity of coordinates in the document input vectors and reduced the SOM computational complexity by several order of magnitude, while providing output identical to the original Kohonen's algorithm. The speed at which our algorithm performed is adequate for real-time interactive applications as long as the number of inputs is less than a thousand (1000), which makes the algorithm a suitable candidate for clustering and visualizing top 200-1000 documents in the rank ordered query results.

Asymptotic complexity of our algorithm is almost *O(S)*, where S is a generalized input size. This implies that under modern CPU speed growth, the interactive applications of SOM can be extended to larger scale tasks (such as clustering entire company intranets or Usenet newsgroups) in the near future.

Maps larger than 100 x 100 still seem to be rather costly to build since they require much computer memory. Since updating the data associated with the map nodes is rather frequent, using slow access storage, such as magnetic disk or a tape drive, would be time prohibitive. Using computer RAM to store this data is so far the only alternative.

Our improvements are independent of other SOM speed-ups, such as parallel implementations and optimizing the algorithm part that finds the winning node at each iteration. In our future research, we will likely to combine such capacities with our approach to achieve better responsiveness and scalability.

7.2 SOM Clustering Abilities

Since our approach used Kohonen's self-organizing map as clustering tool we had to appraise its ability to cluster text documents. We compared SOM against Ward's clustering, a very popular and well tested algorithm. After empirical evaluation, we have concluded that our implementation of Ward's clustering was slightly more precise in detecting associations between documents, but that the performances of these techniques in terms of recall of those associations were not statistically different. This suggests that Kohonen's self-organizing map has clustering abilities close to those of known clustering

techniques. Since the implementation of SOM for text analysis offers several additional valuable features such as providing labels for and visualizing proximity of the clusters it seems to be a viable option for text clustering and categorizing systems.

In order to make the conclusions more general, experiments with different collections seem to be necessary. For example, we are planning a study involving Reuters collection, extensively used in text categorization research. Our empirical evaluation of using SOM as a clustering tool inside an interactive search and visualization system (Adaptive Search) also suggests indirect evidence of its robust clustering abilities, since our evaluation was based on a wide set of search tasks and the entire World Wide Web as the search domain.

We observed that the clustering accuracy is more sensitive to the representation of documents and similarity computation between each pair of documents than to a choice of a clustering algorithm. We suggest exploring semantic relationships between words to improve quality of the output, which may be done based on an existing thesaurus, latent semantic model (Deerwester et al., 1990), or an iterative similarity model (Roussinov, 1998a) that was inspired by this study.

7.3 Customizable SOM

This dissertation and prior studies have established that automatically created maps of concepts improve navigation in large collections of text documents. In addition, we have suggested leveraging navigation by providing interactively the ability to modify the maps

themselves, thus overcoming some of the quality problems associated with unsupervised self-organizing maps. For this purpose we have created and tested a prototype Customizable Self-Organizing Map (CSOM) that builds and refines in real-time a map of concepts found in Web documents returned by a commercial Internet search engine. By observing two search sessions and analyzing trace files from 50 search sessions we have found that users employ and have success with customization features. This study also has suggested that simpler HTML based implementation, which we used in the subsequent empirical study, would require much less learning time.

We also came to the conclusion that implementing a visualization component on a server, different from the one where the search engine is located, requires extensive data exchange between and therefore is not fast enough for interactive applications. A future solution to this problem would be to host the visualization layer at the same server where search engine is, which commercial Internet search service providers should have no problems in implementing. Our solution for the purposes of the experiment was to pre-build maps for the queries used in our experiment.

Our user studies have inspired changes in the navigational paradigm: instead of inspecting clusters one by one, we decided to use clustering as a means of eliciting user feedback and presenting documents in the order of perceived relevance.

Since the entire searching paradigm evaluation described in this dissertation was based on a single empirical study that did not use visual display of self-organizing maps (subjects were able to see only HTML form representation of maps), replicating the results with

graphical interface seems an interesting future direction. For this, current usability of the interface has to be considerably improved. In addition, subjects would be likely to require longer training.

7.4 Adaptive Search: Empirical Study

Based on the statistical evidence and our observations described in the preceding chapter, our suggested approach to information seeking involving self-organizing maps seems promising, even superior to the traditional query based approach. We have compared them on the basis of time users spent searching and the relevance of the documents suggested by the systems. The users found our approach intuitive, easy to use, and preferred using it to the traditional query based paradigm.

We used a WWW searching experiment for our empirical comparison. Since the World Wide Web is a very large and diverse repository, we believe that our results are likely to be generalizable for other large and heterogeneous depositories such as enterprise intranets.

Increasing subject understanding of retrieved results through providing better explanation of why the system selected documents as relevant would seem to be a good future improvement. That might lead users to exhibit more trust in the system's ability not to miss relevant documents, which has been the major impediment to the effectiveness of our approach. We have recorded plenty of user suggestions, implementing which would benefit the usability of our system.

An experiment with a local collection would the reduce number of random factors and would make the output more convincing. At present, we are thinking of experimenting with Usenet newsgroups, Yahoo directory, and Reuters.

Future experiments may involve testing the effects of clustering and term suggestion separately, since in the current study only the combined effect has been quantitatively tested.

The proposed technique also might be compared with simpler competing techniques to account for the detected effects. For example, *Will simply requesting the user to enter more query terms achieve the same effect*? Or, *will suggesting terms only from the query itself but not from the search results have a comparable effect*?

Since our current implementation acts as a layer between a user and a commercial keyword based search engine, the output of user feedback is limited to constructing queries for the engine. In future, we may be able to integrate more tightly the visualizing layer and the retrieval layer to utilize the user feedback not only at the "terms level" but at the "cluster level" as well. New algorithms may be developed to support this interaction.

Our data set consists of more than 40 hours of recorded browsing behavior and hundreds of visited web pages, making it an excellent source for testing future hypotheses related to information seeking models. Improved understanding searching behavior would help developers of information access systems as well as information producers – major contributors to the Web growth. For example, search engine companies already display

ads that are related to perceived user interests. Web developers also are trying to put "catchy" words into their pages as they compete for highly ranked positions in ordered hit lists and thus attract potential consumers.

Future research questions related to information seeking behavior might be:

1. What age and social groups prefer an information navigation paradigm alternative to keyword-based search?

2. What domains seem to be more easily searchable by each technique?

3. How should documents be designed to make them more readily apparent for the users employing information navigation tools?

It is tempting to explore the possibility that the scale of the information depository affected the results obtained in this study. For this, the experiment might be reproduced and the outcomes compared for:

1. *Smaller* collections (100-10000 documents), such as meeting transcripts, service call records, employee manuals, emails.

2. *Intermediate* collections (10000 - 10 million documents), such as newswires, intranets.

3. *Large* collections (10 million and more documents), such as WWW and digital library collections.

Since users' time is likely to be more and more valuable relative to software costs, the concept of information "foraging" (Pirolli et al., 1995) emphasizing the trade-off between the value of information and time spent finding is likely to remain important.

7.5 Dissertation Contributions

7.5.1 Information Clustering Helps Information Seekers

By testing the self-organizing map as a tool for clustering, summarization and eliciting feedback, we have concluded that SOM significantly helps information seekers. This allows us to claim that we have found a way to use of clustering successfully in interactive information access. Other clustering techniques or ways of using them may also turn out to be valuable.

We have suggested an approach that reduces information overload while performing information-seeking tasks. Based on our observations and empirical data, we conclude that we have been able to find a use of *documents clustering that substantially helps interactive information seeking*. Thus, we have answered our central research question.

7.5.2 Contributions to Information Retrieval, Management and Visualization

This study has laid the foundation for future exploration of the Information Navigation paradigm which might enable the user to move smoothly and rationally in the information space. We consider our study only a beginning: Future follow-up studies might investigate the research questions on a deeper level by isolating effects (clustering,

summarization and term suggestion) and providing more impressive results by technical improvements such as explaining system decisions in more detail and providing better usability.

This study supplements studies of information navigation paradigms that researchers in Human Computer Interaction (HCI) have been doing for years. The Information Science community is becoming more and more interested in the interactive approach, to which the call for papers for the next ACM SIGIR conference (Gey et al., 1999) testifies:

> There is a growing opinion in the Information Retrieval community that a key to improving information access systems is to focus attention on the human-computer interface.

This study is one of very few that involve empirical evaluation of new proposed approaches. We not only evaluated the existing technologies, but also suggested a new one. However, the contribution of the study is more than development and testing a working tool; it is an exploration of the paradigm that this tool exemplifies. By building on prior theories and explaining the advantages of this new paradigm offers and why it is more efficient.

We have suggested new metrics for evaluating interacting information access systems suitable for such large depositories as World Wide Web and for a particular scenario when someone is looking for a specific answer. The standard measure of recall is not adequate in such a case, and is not practically computable. To replace the standard measure of precision, we have introduced Answer Rank metrics, in which the position of

a document containing the answer in the ordered list returned by the system determines its degree of relevance.

7.5.3 Contributions to Knowledge Management and Information Systems

From the point of view of management information systems, this study is in knowledge management (KM) area, its technological side. As we indicated in our literature review, the tools for searching, categorizing and visualizing knowledge are vital for successful management and current methods are not adequate for solving the information overload problem. We have suggested and evaluated a technological approach to alleviating information overload, thus paving the way to more successful knowledge management.

Our approach, although designed and tested primarily for Internet users, can be later adopted for searching, categorizing and visualizing company intranets, e-mail repositories and legacy documentation. The metrics we used for the evaluation of the proposed approach are typical in the information systems community and originate from the notions of user productivity and satisfaction. We specifically noted in our experiment the difference between novice and expert users. Our approach targets novice users of the technology, those who do not have enough information science experience to compose efficient boolean queries. From the commercial standpoint, they constitute the majority of consumers of services currently marketed through the Internet e-mail. With the current growth of E-commerce, the ability to attract novice web surfers to companies' products is vital.

Information system researchers traditionally have shared common ground with Human Computer Interaction (HCI) researchers. Our research is interdisciplinary, placed at the border of those two growing and already very influential communities.

Data management researchers in information systems are also becoming increasingly interested in unstructured textual information, in addition to that which is codified in relational and object-oriented databases. We believe that our approach can be extended and adapted to such database applications, in particular searching text fields or building a hierarchy of field names represented in a database.

7.5.4 Possible Social Implications

This study also has several social implications. One of our results is discovery that there exists an answer on the web to even very specific questions as: "Where can I get a good pfeffersteak in Hagerstown, MD USA?" or "How long does it take to get by train from Copenhagen to Oslo?" Also, we established that, armed with the right tools, even novice users of searching technology can unearth those answers.

We consider this study a first step stop toward creating a global self-organizing community of information producers and consumers: the Web of human knowledge. We believe that in many cases the stalemates that frustrate organizations or individuals involve problems that have been studied by someone else. Both parties would enormously benefit from expanded chances of information sharing that helps someone

else find a needed solution. We are happy to suggest our humble step toward creating an information rich society free from the nightmare of information overload.

8. APPENDIX

8.1 Scalable SOM Source Code Fragments

```
/*********************************************************/
/* COPYRIGHT (C) 1996-1999 Arizona Board of Regents      */
/*********************************************************/

/*********************************************************/
/*       AI group                                        */
/*         Department of Management Information Systems  */
/*         College of Business and Public Administration */
/*         The University of Arizona                     */
/*         Tucson, Arizona 85721                         */
/*                                                       */
/*********************************************************/


/*********************************************************/
/* Scalable SOM Algorithm Source Code Fragments
*/

// an abstract object that can be stored to disk or loaded from disk
struct Storable {

  Storable() {}; // should be present to create dummy object before
  // loading from disk
  virtual void operator >> (Stream&) {assert(0); }
  virtual void operator << (Stream&) {assert(0); }

};

// an abstract set template: a set of something
template <class d>
struct SetType : public Storable{

 SetType() { }

 typedef void DataFuncT(d&, void* Data);
 typedef DataFuncT * DataFunc;
 typedef void DataFyncCycleT(d&, long, void* Data);
 typedef DataFyncCycleT* DataFyncCycle;

 virtual void Add(d) = 0;
 virtual void ForAll(DataFunc, void*) = 0;
 virtual void ForAllFixedNumber(DataFyncCycle f, void*, long
CycleNum)=0;
 virtual int total() = 0;
```

```
 // calls f with CycleNum elements of the set sin the order that they
have
 // been added (might repeat elements if neccessary)
};

// a set of something implemented as an array
template <class type>
struct SetAsArray: SetType<type>{


 SetAsArray()
 { buffer = new type[MinSize];pos = 0; limit = MinSize; }
 ~SetAsArray()
 { delete[] buffer; }

 virtual void Add(type v) {expand(); buffer[pos++] = v; }
 virtual void  ForAll(DataFunc f, void* p)
 {
  for(int i=0;i<pos;i++)
      f(buffer[i], p);
 }
 virtual void  ForAllFixedNumber(DataFyncCycle f, void* p, long
CycleNum)
 // calls f with CycleNum elements of the set sin the order that they
have
 // been added
 {
  for(long i=0;i<CycleNum;i++)
    f(buffer[(int)(i % pos)], i, p);
 }

 virtual int total() { return pos; }

 virtual type& operator[] (int i) {
   assert(i>=0 && i<pos);
   return buffer[i];
 }

 virtual void operator >> (Stream& S) { S << pos; S.write(buffer,
pos*sizeof(type)); }

 virtual void operator << (Stream& S) {
   S >> pos; limit = pos; IF(buffer) delete buffer;
   buffer = new type[pos];
   S.read(buffer,pos*sizeof(type));
 }

 protected:

 void expand(){ // checks if there is a need for allocating more memory
  if(pos >= limit){
      type* newBuffer = new type[limit + MinSize];
      for(int i=0;i < limit; i++) newBuffer[i] = buffer[i];
```

```
      delete [] buffer; buffer = newBuffer;
      limit += MinSize;
  }
 }

 enum { MinSize = 100 } ;
      // this is the trick to introduce constants defined sin the sope
of
      // this class. We could write:
      // const int MinSize;
      // But we would need to imitialize MinSize sin each constructor.

 int limit;
 int pos;
 type* buffer;

};

// an abstract Input Vector
/*
  This is the way we treat any input vector. We need just an index,
  distance to given Node, and each coordinate.
*/
struct InputVector: public Storable{

 virtual Number Distance(Node&)=0;
 virtual Number operator[](int)=0;
 virtual LIndex ind()=0;
 virtual int size()=0;
};


inline void InputVectorBoolean::operator >> (Stream& sout) { Data >>
sout; }
inline void InputVectorBoolean::operator << (Stream& sin) { Data <<
sin; }

// input vector with boolean coordinates represented as a sparse
vector:
// a set of non zero indices.

struct InputVectorSparse:
  public InputVector {
// public InputVector, public Storable {


  SetAsArray<int> Data;

  InputVectorSparse(int i) :Ind(i) {  }

  virtual void Add(int x) { Data.Add(x); }
  virtual Number Distance(Node& node);
  virtual int size() { return Data.total(); }
```

```
  virtual LIndex  ind() { return Ind; }
      // returns the index of the input vector

  virtual int& At(int i) { return Data[i]; }
  virtual Number operator[] (int i) { return (Number)Data[i]; }
  virtual void operator >> (Stream& sout) { Data >> sout; }

  virtual void operator << (Stream& sin) { Data << sin; }

protected:
  //  static int instances;
  LIndex Ind;
};

struct InputVectors: public SetType<InputVector*> {
};



// general SOM node
struct Node {

  int y, x; // row, column
  ArrayDynamic<Number> Data;

  Node() { }
  Node(int size, int yi, int xi);

  virtual Number operator[] (int i) { return Data[i]; }
  virtual Number& At(int i) { return Data[i]; }

  virtual Number Value(int i) { return Data[i]; }

  virtual Number Distance(InputVector& );
  virtual void   Adjust(Number Nu, InputVector& V);
  Node(int size, int yi, int xi, FILE* weightFile);
  virtual int size() { return Data.size(); }
};

struct NodeSparse : public Node {

 Number Factor;
 Number SumSquares;

 NodeSparse()
      :Node() { Factor = (Number)1.0; SumSquares = (Number)0.0; }

 NodeSparse(int size, int yi, int xi)
      :Node(size, yi, xi) {

      Reset();
 }
 void Reset(){
```

```
      Factor = (Number)1.0;
      SumSquares = (Number)0.0;
      for (int k=0; k < Data.size(); k++)
        SumSquares += Data[k] * Data[k];
 }

 virtual Number Value(int k) { return Data[k]*Factor; }
 virtual Number operator[] (int i) { return Value(i); }
 virtual Number& At(int i) { return Data[i]; }

 virtual void   Adjust(Number Nu, InputVector& V) {

      // moves this node closer to the input vector sin vector space
      Number s1prime = (Number)0.0;
      Number s2prime = (Number)0.0;

      Number Factor2 = (Number)(1.0 - Nu) * Factor;
      Number FactorSquared = Factor * Factor;
      Number Factor2Squared = Factor2 * Factor2;
      int index;
      Number ai;
      Number newai;
      int length = V.size();
      for (int k=0; k<length; k++) {

        index = (int)V[k];
        ai = Data[index];
        s1prime += ai * ai * FactorSquared;
        newai = ai + (Number)(Nu / ((1.0 - Nu) * Factor));
        Data[index] = newai;
        s2prime += newai * newai* Factor2Squared;
      }

      SumSquares = (Number)((1.0 - Nu) * (1.0 - Nu) * (SumSquares -
s1prime) +
            (Number)s2prime);

      Factor = Factor2;
 }

  NodeSparse(int size, int yi, int xi, FILE* weightFile)
    :Node(size, yi, xi, weightFile) {

      Reset();
  }
};


struct SOM {

 NodeMatrixAbstract * Nodes;
 BuildParam & Params;
 MatrixDynamic<SetAsArray<LIndex> > MappedVectors;
```

```
 MatrixDynamic<SetAsArray<int> > MappedTerms;


 SOM(BuildParam& P);
 ~SOM() { delete Nodes; }

 virtual void Build(InputVectors& V);
 void Write(PString FileName);

 void Train(InputVectors& V);
 void Tune(InputVectors& V);
 void Learn(InputVectors& V, LearnParams& P);

 void MapInput(InputVectors& V);
 virtual void MapOutput()=0;
 void MapOneVector(InputVector& V);

 Node* FindWinningNode(LearnParams& P);
 void  Adjust(int winy, int winx, long cycle, int MaxDist, long
CyclesTotal);
 void  Adjust(Node&, LearnParams& );
 virtual void Print(PString FileName)=0;

 virtual Number NuFunc(int ypos, int xpos, LearnParams& P);

 friend void TrainVector(InputVector& V, void* p);
};



void SOM::Build(InputVectors& V)
{
 ReportLogCT("\nTraining Started.");
 Train(V);

 ReportLogCT("\nTuning Started.");
 Tune(V);

 ReportLogCT("\nMapping Input Started.");
 MapInput(V);

 ReportLogCT("\nMapping Output Started.");
 MapOutput();

 ReportLogCT("\nBuilding SOM Done.");
}


static void LearnVector(InputVector*& V, long cycle, void* p) {
  LearnParams &P=(*(LearnParams*)p);

  // find the winning node ...
  P.CurInp = V;
```

```
  Node* WinningNode = P.Owner->FindWinningNode(P);
  if(csLogWinningNodes) {
    FILE* f = fopen("win.txt", "a+t");
    fprintf(f, "(%d, %d)\n", WinningNode->x, WinningNode->y);
    fclose(f);
  }

  // adjust the neighborhood around the winning node ...
  P.CurCycle = cycle;
  assert(WinningNode);
  P.Owner->Adjust(*WinningNode, P);
}

void DoSom(int _argc, char** _argv)
{

 BuildParam Params;
 getParams(Params);
 char baseName[100];
 char cached[100];
 scanf("%s\n", baseName);
 scanf("%s\n", cached);
 if (strcmp(cached, "true")==0)
   Params.VectorsCached = True;
 else
   Params.VectorsCached = False;


 int nameLen = strlen(baseName)+10;
 char* indexFile = new char[nameLen];
 char* datFile = new char[nameLen];
 char* termFile = new char[nameLen];
 char* outFile = new char[nameLen];
 strcpy(indexFile, baseName);
 strcpy(datFile, baseName);
 strcpy(termFile, baseName);
 strcpy(outFile, baseName);
 strcat(indexFile, ".ind");
 strcat(datFile, ".dat");
 strcat(termFile, ".terms");
 strcat(outFile, ".map");


 SomSparseTextualDocuments som(datFile, Params,  termFile);
 som.Print(outFile);
}
```

8.2 Ward's Clustering Source Code Fragments

```
/*******************************************************/
/* COPYRIGHT (C) 1996-1999 Arizona Board of Regents    */
/*******************************************************/
```

```
/*******************************************************/
/*       AI group                                      */
/*         Department of Management Information Systems  */
/*         College of Business and Public Administration */
/*         The University of Arizona                     */
/*         Tucson, Arizona 85721                         */
/*                                                       */
/*******************************************************/


/*******************************************************/
/* Ward's Clustering
*/



class ClusterAbstract{

public:
 virtual Number Distance(const ClusterAbstract& c) { return 0.; }
 // This should be overwritten in derived classes
 // For example, for points in vector space, this should be
 // Euclidean distance

 virtual int Total() const { return 0; }
 // returns the total number of objects in this cluster

 virtual void Print(FILE* ) { }
 // prints IDs of all objects merged into this one

 typedef void (*ClusterAbstractFunc)(ClusterAbstract& , void*);
 typedef Boolean (*BooleClusterAbstractFunc)(ClusterAbstract& , void*);

 virtual void ForAll(ClusterAbstractFunc, void*) { assert(False); }
 virtual ClusterAbstract* FindP(BooleClusterAbstractFunc, void*)
 { assert(False); return NULL;}
 // calls 'ClusterAbstractFunc' with each object in this cluster

};


// a node in a binary tree
template <class NodeData>
// This is a template, so any data specified 'NodeData' by types can be
stored
// in each node
class TreeNode {

 public:
 TreeNode(NodeData  nd, TreeNode* l = NULL, TreeNode* r = NULL)
  :Data(nd) { Left = l; Right = r; id = count ++;}
```

```
 NodeData  Data; // This depends on the partiluar use of the
      // template. For example, it can be a pointer to some data stored
in
      // each node of the tree.
 TreeNode* Left; // children nodes
 TreeNode* Right;
 int          id; // for convenience of debugging and printing
 int static count; // for convenience  in assigning 'id'
      // 'static' means same variable for all instances of this class
};


// a binary tree

template <class NodeData>
// This is a template, so any data types can be stored in each node

class Tree {
 public:
 Tree() { root = NULL; }

 typedef TreeNode<NodeData> Node; // for convenience

 void Print(const char* fileName);
 static  void TraversePreOrder(Node& n, void f(Node&, void*), //
      void* ExtraData)
 {
  assert(&n);

  f(n, ExtraData); // call 'f' with the parent node

  if(n.Left){
      assert(n.Right);

      TraversePreOrder(*n.Left, f, ExtraData); // visit left subtree
      TraversePreOrder(*n.Right, f, ExtraData); // visit right subtree
  }
 }

 static void TraversePreOrderC(Node& n,
      Boolean f(Node&, void* ExtraData),
      void* ExtraData)
 {
  assert(&n);

  if(!f(n, ExtraData)) // call 'f' with the parent node
   if(n.Left){
      assert(n.Right);

      TraversePreOrderC(*n.Left, f, ExtraData); // visit left subtree
      TraversePreOrderC(*n.Right, f, ExtraData); // visit right subtree
   }
 }
```

```
 void setRoot(Node& n) { assert(root == NULL); root = &n; }
 // Sets the root of the tree. Should be called only once.
 Node& Root() { return *root; }
 // Sets the root of the tree. Should be called only once.

 protected:
 Node* root;
};


struct TaggedValue{
 int tag;
 Number value;
 TaggedValue(int t, Number n = 0){tag = t; value = n; }
 TaggedValue() { value = 0.; }
 virtual ~TaggedValue() { }
 Boolean operator == (const TaggedValue& t)
      { return tag == t.tag && value == t.value; }
};

class PointSparse: public PointAbstract,
      public CoordinateListOrdered{

 public:
       PointSparse() { }
 virtual Number operator [](int i) { assert(False); return 0.; }
 virtual void ForAllNonZero(void f( CoordPair*& , void*),
      void* p)const { ForAll(f, p); }

};

// Implementation of Ward's clustering on points. It creates a
ClusterTree

class WardClustering{

 public:

 WardClustering(const PointList& List, ClusterTree& , int
NumberOfClustersRequired = 1);

 void Print(const char* File);

 protected:

 ClusterList CL;
};


Number VarianceEstimate(ClusterInTree& c)
{
```

```
 sum = 0;
 CurCenterPtr = (ClusterPoint*)&c;
 void AddSquareDist(ClusterNode& n, void* );
 Tree<ClusterInTree&>::TraversePreOrder(c.treeNode(), AddSquareDist,
NULL);

 Number r;

 if(c.Total()>1)
       r = sum / ((Number)c.Total()-1);
 else
       r = 0;

 return r;
}

// used for temporary data passed to various functions
struct PairCoordData{
 const ClusterPoint *p1,*p2;
 ClusterPoint *p3;
};

void IterateOrderedLists(
      const ListLP<TaggedValue*>& List1,
      const ListLP<TaggedValue*>& List2,
   void f(TaggedValue* , TaggedValue* , void * ExtraData),
   void* ExtraData);



void IterateOrderedLists(
      const ListLP<TaggedValue*>& List1,
      const ListLP<TaggedValue*>& List2,
      void f(TaggedValue* , TaggedValue* , void * ExtraData),
      void* ExtraData)
{

#define MAXINT (1 << (sizeof(int)*8-2))
#define MaxInt MAXINT
#define Tag(p, L) p ? L.At(p)->tag : MaxInt

#define move1() \
  tmp1 = List1.RightAfter(tmp1); \
  tag1 = Tag(tmp1, List1);

#define move2() \
  tmp2 = List2.RightAfter(tmp2); \
  tag2 = Tag(tmp2, List2);

 LLNode<TaggedValue*>* tmp1 = List1.head();
 LLNode<TaggedValue*>* tmp2 = List2.head();
 int tag1 = Tag(tmp1, List1);
 int tag2 = Tag(tmp2, List2);
```

```
 while(tmp1 || tmp2){

  if(tag1 > tag2){
   f(&TaggedValue(tag2), List2.At(tmp2), ExtraData);
      move2();
  }
  else if(tag2 > tag1){
   f(List1.At(tmp1), &TaggedValue(tag1), ExtraData);
      move1();
  }
  else{

   assert(tag1 == tag2);
   assert(tmp1 && tmp2);
   f(List1.At(tmp1), List2.At(tmp2), ExtraData);
   move1();
   move2();
  }
  IterateCount++;
 }
}

ClusterNodePtr Nearest(ClusterNodePtr c, const ClusterList& List)
// returns the nearest neighbor cluster in the given set
// (as the one with the smallest distance)
{
 S1 D;
 D.minDistance = 1e10;
 D.target = List.At(c);
 D.exclude = c;  // itself should not be considered
 D.nearest = NULL;
 ClusterNodePtr tmp = List.head();
 while(tmp){

  Check(tmp, List, D);
  tmp = List.RightAfter(tmp);
 }
 return D.nearest;
}

WardClustering::WardClustering(const PointList& List,
      ClusterTree& T, int NumberOfClustersRequired)
{
 Boolean AlreadyHadEnoughManyClusters = False;
 TreeNode<ClusterInTree&>::count = 1;
 CL = List;


 Boolean done;
 int NumberOfClusters = List.total(); assert(NumberOfClusters);
 ClusterPoint* LastMerged = NULL;
 ClusterNodePtr currentClusterPtr = CL.head();
```

```
ClusterNodePtr nearestNeighborPtr = NULL;
ClusterNodePtr oneBefore = NULL;
int step = 0;
remove(csLogFile);

do{
     step++;

  int ChainLength = 0;
  Boolean Found = False;
  while(!Found){


   nearestNeighborPtr = Nearest(currentClusterPtr, CL);
   assert(nearestNeighborPtr != currentClusterPtr);
   Found = (Nearest(nearestNeighborPtr, CL) == currentClusterPtr);
   ChainLength++;
      assert(ChainLength < CL.total()); // should not cycle!

      if(!Found){
       oneBefore = currentClusterPtr;
       currentClusterPtr = nearestNeighborPtr;
      }
     }
     assert(Found);
     ClusterPoint* currentCluster = CL.At(currentClusterPtr);
     ClusterPoint* nearestNeighbor = CL.At(nearestNeighborPtr);

     LastMerged = new ClusterPoint(*currentCluster, *nearestNeighbor);
     CL.append(LastMerged);
     CL.AdvanceRemove(currentClusterPtr);
     CL.AdvanceRemove(nearestNeighborPtr);

     if(LogFlag){

      FILE* f = fopen(csLogFile, "at+");
      fprintf(f, "\n\nStep %d. The cluster:", step);
      currentCluster->Print(f);
      fprintf(f, " and the cluster:");
      nearestNeighbor->Print(f);
      fprintf(f, " are merged to produce the cluster:");
      LastMerged->Print(f);
      fclose(f);
     }

  NumberOfClusters--;
  assert(NumberOfClusters == CL.total());

  int Num = NumberOfNormalClusters(CL, 5);
  if(!AlreadyHadEnoughManyClusters)
   AlreadyHadEnoughManyClusters = (Num >= NumberOfClustersRequired);
  done = AlreadyHadEnoughManyClusters &&
           (Num < NumberOfClustersRequired);
```

```
   done = (NumberOfClusters == 1);

   if(ChainLength > 2)
            currentClusterPtr = oneBefore;
   else
            currentClusterPtr = CL.head();

 } while(!done);

 assert(LastMerged);
 T.setRoot(LastMerged->treeNode());
}
```

8.3 Customizable SOM Source Code Fragments

```
/*********************************************************/
/* COPYRIGHT (C) 1996-1999 Arizona Board of Regents      */
/*********************************************************/

/*********************************************************/
/*       AI group                                        */
/*         Department of Management Information Systems  */
/*         College of Business and Public Administration */
/*         The University of Arizona                     */
/*         Tucson, Arizona 85721                         */
/*                                                       */
/*********************************************************/


/*********************************************************/
/* Interaction with AltaVista Search Engine
*/

void StoreSummaries(PStr PageFile, HitList & L)
{
      FStream S((char*)PageFile);
      char* bu  = new char [S.getSize()];
      S.read(bu, S.getSize());
      CStream SM(bu);


      LexemmStream h(SM);

      while(!h.end()){


            if(h.LastPatternIs(" </b><a href=\"")){ //
             Pattern Link = h.getFromUntil(h.getPos(), '"');
             assert(Link.getSize() > 4);
             // extract title
             Pattern Title = TextBefore(h, "</b>", True); //
```

```
             // extract summary
             Pattern summary = TextBefore(h, "<br><b>URL:");
             assert(summary.getSize() > 0);

             // remove special symbols
             summary = SafePattern(summary);

             HitInfo H(Link, Title, summary);
             L.append(H);

            }
            if(!h.end())
                  h.getC();
      }
}


Pattern AltaQuery(int StartPage, Query& query,
      Boolean Advanced = True, char* rank = NULL)
{
 char bu[1000];

 if(Advanced){
  if(mode == news)
   sprintf(bu, "http://www.altavista.digital.com/cgi-
bin/query?pg=aq&what=news&stq=%d&q=%s",
   StartPage, WebSyntax(query).getString());
  else
  sprintf(bu, "http://www.altavista.digital.com/cgi-
bin/query?pg=aq&stq=%d",
   StartPage); //

  return Pattern(bu) + "&q=" + WebSyntax(query) + "&r=" +
WebSyntax(Query(rank));
 }
 else{
  if(mode == news)
   sprintf(bu, "http://www.altavista.digital.com/cgi-
bin/query?pg=q&what=news&stq=%d&q=%s",
   StartPage, WebSyntax(query).getString());
  else{
  if(rank)
   sprintf(bu, "http://www.altavista.digital.com/cgi-
bin/query?pg=q&kl=en&stq=%d&q=%s&r=%s",
   StartPage, WebSyntax(query).getString(), rank); //
  else
   sprintf(bu, "http://www.altavista.digital.com/cgi-
bin/query?pg=q&kl=en&stq=%d&q=%s",
   StartPage, WebSyntax(query).getString());
  }
 }
 return Pattern(bu);
}
```

```
void QueryAltaVista(URLList & Hits, Query& query,
 Boolean downloadSummary = False, Boolean Advanced = True,
 HitList* HLP = NULL, char* rank = NULL) // RECENT

// gets all pages with hits and get URLs out of them
{
 URLList URLbuffer; // RECENT
 HitList   HitsBuffer;

 // form a query for the first page
 URL queryURL = AltaQuery(0, query, Advanced, rank);
 // csAltaVistaPrefix + WebSyntax(query)

 // ask for the first page
 const char* csTempAltaOutputFileName = "alta.htm";
 GetPage(queryURL, csTempAltaOutputFileName);

 // get number of pages from the first page ...
 int NumHits = NumberOfHits(csTempAltaOutputFileName);
 char bu[100];
 sprintf(bu, "\nFound hits:%d", NumHits);
 ReportLog(bu);

 int csMinInPage = 10;
 if(mode == news)
  csMinInPage = 30;


 int MaxNumPages = NumHits / csMinInPage + 1;


 NoMore(NumHits, maxNumberOfHits); //

 int MaxAVPagesToParse = 20;

 int checked = 0;
 Boolean stop = False;
 int EmptyCount = 0;

 if(DownloadSummaries || downloadSummary)
       remove("summaries.txt");

 //
 // pre-fetch pages using Java program


 FILE* f = fopen("tofetch.txt", "wt");
 for(int i=0; i<min(MaxAVPagesToParse, MaxNumPages); i++){

  URL addr = AltaQuery(i*csMinInPage, query, Advanced, rank);
  fprintf(f, "%s\n", addr.getString());
```

```
}
fclose(f);

extern Boolean UsePageCache;
if(UsePageCache){

 if(!ToFetchCached("tofetch.txt")){
  system0(Pattern(javaStr) +  " GetUrls");
  AddFetched("tofetch.txt");
 }
}
else
  system0(Pattern(javaStr) +  " GetUrls");

countSummary = 0;

NoMore(MaxNumPages, MaxAVPagesToParse); //
while((Hits.total() < NumHits) && (checked < MaxNumPages) && !stop){

 // get hits from the page
 URL addr = AltaQuery(checked*csMinInPage, query, Advanced, rank);

 sprintf(bu, "fetched%d.html", checked);
 remove("homepage1.link");
 extern const char* csTmpPage;
 remove(csTmpPage); //
 FileCopy(bu, csTmpPage);
 int res  = ParseFetchedURL();

 checked ++;

 if(DownloadSummaries || downloadSummary){

       HitList   hitList;
       extern const char* csTmpPage;

       StoreSummaries(csTmpPage, hitList);
       StoreSummaries(csTmpPage, HitsBuffer);

     if(DownloadSummaries || downloadSummary) //
        hitList.Store("summaries.txt");


     hitList.ForAll(StoreURL, &URLbuffer); //
       if(HLP)
      hitList.ForAll(StoreHitInfo, HLP); //

     sprintf(bu, "\nFound links from the page:%d", hitList.total());
       ReportLog(bu);
       assert(hitList.total());
 }
}
Hits = URLbuffer;
```

```
 FStream S("hits.bin", "wb");
 HitsBuffer >> S;
}

import java.io.*;
import java.net.*;
import java.util.*;

public class GetUrlsLarge implements Runnable {

  public static int BUFFER_SIZE = 1024;
  public static int NumberOfServers = 200; // much faster

  Vector Task;
  static Integer fetched = new Integer(0);
  int NumBlocks = 1;
  int Index;
  static int HowMany = 0;

  protected void finalize(){

      try{
        File localFile = new File("num_of_hits.txt");
        OutputStream out = new FileOutputStream( localFile );
        (new PrintStream(out)).println(Task.size());
        out.close();
      }
    catch(Exception e) {
      System.err.println(e);
    }

  }


  public GetUrlsLarge (Vector TaskInit, int N, int I)
  { Task = TaskInit;NumBlocks = N; Index = I; HowMany++; }

  public void run() {

    Thread thread = Thread.currentThread();


      StringBuffer UrlString = new StringBuffer();
      StringBuffer FileName = new StringBuffer();

      boolean stop = false;
      while(!stop){

       thread.yield();

       synchronized(fetched){
```

```
        if(fetched.intValue() < Task.size()){


UrlString.append((String)Task.elementAt(fetched.intValue()));
             FileName.append("fetched" + (new
Integer(fetched.intValue())).toString() + ".html");
             fetched  = new Integer(fetched.intValue() + 1);
        }
        else
            stop = true;
       }

       if(stop)
                  continue;


     InputStream in = null;
     OutputStream out = null;

       try{

        String urlString = UrlString.toString();
        File localFile = new File(new String(FileName));

        UrlString.setLength(0);
        FileName.setLength(0);

        out = new FileOutputStream( localFile );
      URL url = new URL(urlString);
      in = url.openStream();


      byte[] buffer = new byte[BUFFER_SIZE];

        int bytes_read;

        int count = 0;
        int total_read = 0;
        int SizeToRead = NumBlocks*BUFFER_SIZE;

      while (((bytes_read = in.read(buffer)) != -1) && (total_read <
SizeToRead)){

            out.write(buffer, 0, bytes_read);
            total_read += bytes_read;
            thread.yield();

        }
       }
     catch(Exception e) {
      System.err.println(e);
     }
     finally { // always close the streams, no matter what.
```

```
      try { in.close(); out.close(); } catch (Exception  e) {}
     }
     thread.yield();
      }
  }

 public static void main(String[] args) {

  try{

      int NumBlocks = Integer.parseInt(args[0]);

      File localFile = new File("tofetch.txt");
    InputStream is = new FileInputStream(localFile);
      DataInputStream in = new DataInputStream(is);
      String s;
      int count = 0;
      Vector Task = new Vector ();

      while((s = in.readLine()) != null){

            if((new String(s)).length() > 3)
                        Task.addElement(s);
      }


      Vector ThreadVector = new Vector();
      for(int i=0; i<Math.min(NumberOfServers, Task.size()); i++){

       Thread t = new Thread(new GetUrlsLarge(Task, NumBlocks, i ));
       ThreadVector.addElement(t);
      }
      for(int i=0; i<NumberOfServers; i++){ //
            ((Thread)ThreadVector.elementAt(i)).start();

      }
  }
  catch(Exception e) {

      System.err.println("Problems while starting fetching ...");
  }
 }
}
```

8.4 Adaptive Search Source Code Fragments

```
/*****************************************************/
/* COPYRIGHT (C) 1996-1999 Arizona Board of Regents      */
/*****************************************************/

/*****************************************************/
/*       AI group                                    */
/*         Department of Management Information Systems  */
```

```
/*         College of Business and Public Administration */
/*         The University of Arizona                     */
/*         Tucson, Arizona 85721                         */
/*                                                       */
/*********************************************************/


/*********************************************************/
/* Adaptive Search Source Code Fragments
*/

/*
Decreases term coordinate if a phrase contaning this term is also in
the same
document.
  */
void FilterSubsumtion(PStr name, PStr oname,  PStr terms, int Size,
int& total)
{
 // finds all subsumtions
 TermArray Terms; TermID Total;
 ReadTerms(Terms, terms, Total);


 ArrayDynamic<ListLO<int> > SubsumedList(Size);
 for(unsigned int i=0;i<Total;i++){ //

      for(unsigned int j=0;j<Total;j++){ //

      assert(Terms[j]);
      assert(Terms[i]);
        if(i!=j)
         if(strstr(Terms[j], " ") != NULL){ // a phrase //
            char* start = strstr(Terms[j], Terms[i]);
            if(start){
             int l = strlen0(Terms[i].getString());
             if(start[l] == ' ' || start[l] == 0)
                  SubsumedList[i].append(j);
            }
         }
       }

 }

 // does a file re-write
 FILE* f = fopen(name, "rt");
 FILE* fo = fopen(oname, "wt");
 ArrayDynamic<Number> Coord(Size);
 total = 0;
 while(!feof(f)){

  // read coordinates
  for(int i=0;i<Size;i++)
```

```
      Coord[i] = 0;

  int id, num;
  check(fscanf(f, "#%d %d\n", &id, &num) == 2);
  total ++;

  for(i=0;i<num;i++){

      int axis;
      Number value;
      check(fscanf(f, "%d %e\n", &axis, &value) == 2);
    if(value > 0)
        Coord[axis] = value;
  }

  // adjust coordinates
  for(i=0;i<Size;i++) if(Coord[i] > 0.001)
   for(int j=0;j<Size;j++) if(Coord[j] > 0.001)
         if(SubsumedList[i].Find(Same, &j))
             Coord[i] -= Coord[j];

  // print coordinates
  int count = 0;
  for(i=0;i<Size;i++) if(Coord[i] > 0) count++;
  fprintf(fo, "#%d %d\n", id, count);
  for(i=0;i<Size;i++) if(Coord[i] > 0)
      fprintf(fo, "%d %e\n", i, Coord[i]);


 }
 fclose(f);
 fclose(fo);
}


void PutTopicsStatus(Boolean extras = True) // as form
// preserves checked/uncheked concepts
{
 if(extras) //
  fprintf(out, "\
<dl> \
    <dt>Select &quot;+&quot; for relevant terms, &quot;-&quot; \
        for irrelevant ones, or just ignore them.</dt> \
</dl>");

 fprintf(out, "<font size=\"1\">");
 fprintf(out, "<table border=\"0\" >");

 count = 0;
 SubCount = 0;

 if(!extras)
  Status::Regions.ForAll(StoreLabels, NULL);
```

```
 else
  Status::Regions.ForAll(Store, NULL);

 fprintf(out, "</font></table>");

 if(extras && !Status::isDemo){ //
  fprintf(out, "<p>You can add other relevant words or phrases (in
quotes):");

  fprintf(out, Pattern("<p>\
    <textarea \
    name=\"aq\" rows=\"1\" cols=\"55\" maxlength=\"800\"
wrap=\"virtual\">") +
      Status::extraQuery + "</textarea>");
 }
 if(Status::isDemo){ // put feedback button
      PutFile("middle_e.html");
 }
}


void StoreHTML(HitInfo& h, void* )
{
 // store title
 fprintf(out, "<DD><B><FONT SIZE=-1></FONT>");

 if(h.Title.getString()){



  fprintf(out, "<b>%d. </b><A HREF=\"", count++);
  fprintf(out, "%s\">%s  </A></B>  ", h.URL.getString(),
h.Title.getString());
 }

 // store link
 if(h.URL.getString())
  fprintf(out, "<I><A HREF=\"%s\">%s</A></I></DT>", h.URL.getString(),
h.URL.getString());

 // store summary
 if(h.Summary.getString())
  fprintf(out, "<DD>%s<BR><BR></DD>", h.Summary.getString());
}


// puts links and summaries from Alta-Vista results page
void PutHits(int PageIndex)
{
// update "alta.htm"
 Query q = Status::query;
 URL queryURL = AltaQuery(PageIndex*10, q, Status::Advanced);
 const char* csTempAltaOutputFileName = "alta.htm";
```

```
 GetPage(queryURL, csTempAltaOutputFileName);


        HitList   hitList;
        fprintf(out, "<P>About %d documents match your query.",
NumberOfHits("alta.htm"));
        StoreSummaries("alta.htm", hitList);
      count = PageIndex*10+1;
        fprintf(out, "<DL>");
        hitList.ForAll(StoreHTML, NULL);
        fprintf(out, "<BR> </DL>");
}

// from the pre-created list
void PutHitsRanked(int PageIndex)
{
 if(Status::isDemo)
  PutFile("hits_explain.html");

        FStream S(csRankedFile, "rb");
        HitList hitList;
        hitList << S;

        int start = PageIndex*10;
      count = start+1;
        fprintf(out, "<DL>");
        int QueryCount = 0;
        int NextGroupStart = 0;

        for(int i=0;i< min(hitList.total(), start+10); i++){

              int Num;
              if(i == NextGroupStart) do{

                  Pattern Query = Status::QueryTrace[QueryCount];

                  char* NumStr = strstr(Query.getString(), foudnStr);
                  IF(NumStr){
                   NumStr += strlen0(foudnStr);
                 Num = atoi(NumStr);
                   QueryCount ++;
                   NextGroupStart = i + Num;

                   if(i >= start ){
                    fprintf(out, "<font color=\"#000080\"
size=\"1\">"); // format
                     fprintf(out, "%s<P>", Query.getString() ); // text
                     fprintf(out, "</font>"); // restore format
                   }
                  }
              }
              while(Num == 0);
```

```
              if(i >= start ) //
               StoreHTML(hitList[i], NULL);
        }
        fprintf(out, "<BR> </DL>");

        PutLinkNextPrevEx(PageIndex, ::URLforPageR, 200); //
}

Pattern URLforPage(int i)
{
 return CGIurl + "?formname=display&start=" + NumberStr(i);
}
Pattern URLforPageR(int i)
{
 return CGIurl + "?formname=rank&start=" + NumberStr(i);
}

void PutLinkNextPrevEx(int PageIndex, Pattern URLforPage(int ), int
TotalDocs)
{
 int NumHits = TotalDocs;
 Pattern URLCallPrev;

 fprintf(out, "<font face=arial size=-1>Pages:");
 if(PageIndex-1 >= 0)
  fprintf(out, "<a href=\"%s\">[<b>&lt;&lt;</b>]</a>",
URLforPage(PageIndex-1)());

 int MaxHitShow  = min(csMaxHitShow, (NumHits+csInPage-1)  / csInPage);

 for(int i=0;i<MaxHitShow;i++){

  if(i != PageIndex)
   fprintf(out, "<a href=\"%s\"> %d</a>", URLforPage(i)(), i+1);
  else
   fprintf(out, "<b> %d</b>", i+1, URLforPage(i)());
 }
 if(PageIndex+1 < MaxHitShow)
  fprintf(out, "<a href=\"%s\"> [<b>&gt;&gt;</b>]</a> <P>",
URLforPage(PageIndex+1)());

 fprintf(out, "</font>");
}

// handles user query
void HandleQuery(char* buffer)
{

 // get parameters
 Query query = GetFieldData(buffer, "q");
 FromInternet(query);  //
 ReportLog("\nHandling query: " + query); //
```

```
Boolean Advanced = (GetFieldData(buffer, "pg") == "aq");

Boolean FullDocs = (GetFieldData(buffer, "full") == "yes");
URLList Hits;
int total = 10;

Status::Reset(); //
Status::query = query;
Status::Advanced = Advanced;

 //
if(GetFieldData(buffer, "concepts") == "no"){

      Status::Regions.clear();
      HandleRank(buffer);
      return;
}


if(GetFieldData(buffer, "download") != "no"){

 remove(TextInput); //
 Status::Reset();
 total  = DownloadSummaries(query, Hits, Advanced, FullDocs);
 Status::TotalDocs = total;
 Status::FullDocumentsLoaded = False; // to keep consistent
 Status::TotalDocsMatching = NumberOfHits("alta.htm");
}

//
Boolean WaitUntilFull = !GetFieldData(buffer, "wait").isEmpty();
if(WaitUntilFull){

 DownloadHitsParallel(Hits, WaitUntilFull); //
 HandleRefine((Pattern(buffer) + "&full=yes").getString());
 AccomodateNewInput(buffer);
 Pattern dir = GetFieldData(buffer, "d");
 if(!dir.isEmpty()){

       Status::Print("status.txt");
       FileCopyDS("", "status.txt", dir);
       FileCopyDS("", "summaries.txt", dir);
 }
 return;
}


SomParams.VectorSize = min(50, total);


SomParams.GridHeight = 10;
SomParams.GridWidth = 10;
```

```
 if(GetFieldData(buffer, "map") != "no")
  CreateMap(GetFieldData(buffer, "ind") == "no");



 // for experiment, no applet shows up
 //
 if(GetFieldData(buffer, "applet") == "no") //  fix
  UseApplet = False;


 //
 if(GetFieldData(buffer, "experiment") == "yes")
       PutCurrentE();
 else
       PutCurrent();

 if(FullDocs) //
   DownloadHitsParallel(Hits); //

}
```

REFERENCES

Ba, S.L., Lang, K.R., & Whinston, A.B. (1997). Enterprise decision support using intranet technology. *Decision Support Systems*, 20(2) (pp. 99-134), June 1997.

Bannan, J. (1997). Intranet document management. Addison-Wesley Developers Press, Reading, MA.

Bartell, B.T., Cottrell, G.W., & Belew, R.K. (1995). Representing documents using an explicit model of their similarities. *Journal of the American Society for Information Science*, 46(4) (pp. 254-271).

Bates, M. J. Subject Access in Online Catalogs: A Design Model (1986). *Journal of the American Society for Information Science*, 37(6) (pp. 357-376).

Blair, D.C. & Maron, M.E. (1985). An Evaluation of Retrieval Effectiveness for a Full-Text Document-Retrieval System. *Communications of the ACM*, 28(3) (pp. 289-299).

Bowman, C.M. (1994). The Harvest information discovery and access system. *Proceedings of the Second International World Wide Web Conference '94*, October 17-20, 1994, Chicago, IL.

Card, S.K., Robertson, G.G., & York, W. (1996). The WebBook and the Web Forager: An Information Workspace for the World-Wide Web. *Proceedings of the ACM/SIGCHI Conference on Human Factors in Computing Systems* (pp. 111-119). Vancouver.

Carroll, J.M., & Thomas, J.C. Fun. (1988). *SIGCHI Bulletin*, January 1988, volume 19, number 3.

Chen, H., & Dhar, V. (1990). User misconceptions of online information retrieval systems. *International Journal of Man-Machine Studies*, 32(6) (pp. 673-692).

Chen, H., & Lynch, K.J. (1992). Automatic Construction of Networks of Concepts Characterizing Document Databases. IEEE Transactions on Systems, Man, and Cybernetics, 22(5) (pp. 885-902).

Chen, H., Hsu, P., Orwig, R., Hoopes, L., & Nunamaker, J.F. (1994). Automatic concept classification of text from electronic meetings. Communications of the ACM, 37(10) (pp. 56-73), October 1994.

Chen, H., Schuffels, C., & Orwig, R. (1996). Internet Categorization and Search: A Self-Organizing Approach. *Journal of Visual Communication and Image Representation*, 7(1) (pp. 88-102).

Chen, H., Schatz, B.R., Ng, T.D., Martinez, J.P., Kirchhoff, A.J., & Lin, C. (1996a). A parallel computing approach to creating engineering concept spaces for semantic retrieval: The Illinois Digital Library Initiative Project. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(8), (pp. 771-782), August 1996.

Chen, H., Houston, A., Yen, J., & Nunamaker, J.F. (1996b). Toward intelligent meeting agents. *IEEE COMPUTER*, 29(8), (pp. 62-70), August 1996.

Croft, W.B. (1995). Clustering large files of documents using single link method. *Journal of the American Society for Information Science*, 28 (pp. 341-344).

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

Davenport, T.H., & Prusak, L. (1998). Working Knowledge: How Organizations Manage What They Know. *Harvard Business School Press*, Boston, MA.

DeBra, P. & Post, R. (1994). Information retrieval in the World-Wide Web: Making client-based searching feasible. *Proceedings of the First International World Wide Web Conference '94*, Geneva, Switzerland.

Deerwester S., Dumais S., Furnas G., Landauer T.K., & Harshman R., Indexing by Latent Semantic Analysis. *Journal of the American Society for Information Science* 41 (1990), 391-407.

El-Hamdouchi, A., & Willett, P. (1986). Hierarchical document clustering using Ward's method. *Proceedings of the 9th International Conference on Research and Development in Information Retrieval* (pp. 149-156). Washington.

Ericsson K.A., & Simon H.A. (1993). Protocol analysis: verbal reports as data. MIT Press, Cambridge, Mass.

Everitt, B.S. (1974). *Cluster Analysis*. New York. John Wiley & Sons, Inc.

Furnas, G. W., Landauer, T. K., Gomez, L. M., & Dumais, S. T. (1987). The Vocabulary Problem in Human-System Communication. *Communications of the ACM*, 30(11) (pp. 964-971).

Gey, F., Hearst, N, & Tong, R. (1999). Call for paper: 22nd International ACM SIGIR Conference on Research and Development in Information Retrieval. *Communications of the ACM*, 42(5).

Gordon. M. (1997). It's 10 a.m. Do You Know Where Your Documents Are? The Nature and Scope of Information Retrieval Problems in Business. *Information Processing and Management*, 33(1) (pp. 107-121).

Grefenstette, G. (1994). *Explorations in Automatic Thesaurus Discovery*. Kluwer Academic Publishers, Boston, MA, 1994.

Hansen, M.T., Noria N., & Tierney T. (1999). What's your strategy for managing knowledge? Harvard Business Review, 77(2) (pp. 106-).

Hearst, M.A. (1997). Interfaces for Searching the Web. *Scientific American*, March (pp. 68-72).

Hearst, M.A. (1999). User Interfaces and Visualization, in *Modern Information Retrieval*, edited by Ricardo Baeza-Yates and Berthier Ribeiro-Neto, Addison-Wesley Publishing Company.

Hearst, M.A. (1999a). The Use of Categories and Clusters in Information Access Interfaces, in Natural Language Information Retrieval, Strzalkowski (ed.), Kluwer Academic Publishers, 1999.

Hearst, M.A., & Pedersen, J.O. (1996). Reexamining the Cluster Hypothesis: Scatter / Gather on Retrieval Results. *Proceedings of the Nineteenth Annual International ACM Conference on Research and Development in Information Retrieval* (pp. 76-84). Zurich.

Honkela, T., Kaski, S., Lagus, K., & Kohonen, T. (1996). Newsgroup exploration with WEBSOM method and browsing interface. In Report A32, Helsinki University of Technology, January 1996.

Ingwersen, P. (1994). Polyrepresentation of Information Needs and Semantic Entities: Elements of a cognitive theory for information retrieval interaction. *Proceedings of the International ACM SIGIR'94 Conference*, (pp. 101-110).

J. Ward, (1963). Hierarchical grouping to optimize an objection function. *Journal of the American Statistical Association*, 58 (pp. 236-244).

Jain, A.K., & Dubes, R.C. *Algorithms for Clustering Data.* 1988. Prentice Hall.

Jardine, N., & van Rijsbergen, C.J. (1971). The Use of Hierarchic Clustering in Information Retrieval. *Information Storage and Retrieval*, 7 (pp. 217--240).

Kaski, S., Honkela, T., Lagus, K., & Kohonen, T.(1998). WEBSOM--self-organizing maps of document collections. *Neurocomputing*, 21 (pp. 101-117).

Kohonen, T. (1989). *Self-Organization and Associative Memory*. Springer Verlag.

Kohonen, T. (1995). *Self-Organizing Maps*. Berlin, Heidelberg. Springer-Verlag.

Lagergren, E., & Over, P. (1998). Comparing interactive information retrieval systems across sites: The trec-6 interactive track matrix experiment. In Proceedings of the *21st Annual International ACM/SIGIR Conference* (pp. 164-172).

Landauer, T.K., Egan, D.E., Remde, J.R., Lesk, M., Lochbaum, C.C., & Ketchum, D. (1993). Enhancing the usability of text through computer delivery and formative evaluation: the superbook project. *In Hypertext: A psychological perspective,* ed. by C. McKnight, A. Dillon, & J. Richardson (pp. 71-136). Ellis Horwood.

Lawrence, S. (1999). Search Engines Biased, Out-Of-Date, And Index No More Than 16% Of The Web. Science Daily, July 12, 1999, http://www.sciencedaily.com/releases/1999/07/990712075603.htm

Leouski, A., & Allan, J. (1998). Visual Interactions with a Multidimensional Ranked List, *Proceedings of the Twenty First Annual International ACM Conference on Research and Development in Information Retrieval,* (pp. 353-354), Melbourne, Australia.

Murtagh, F. (1985). *Multidimensional Clustering Algorithm.* Vienna. Physica-Verlag.

Nonaka, I. (1994). A dynamic theory of organizational knowledge creation. *Organization Science,* 5(1) (pp. 14-37).

Nunamaker, J.F., Chen H., & Purdin, T.D.M. (1991). Systems development in information systems research. *Journal of Management Information Systems,* 7(3) (pp. 89-106).

Nunamaker, J. F., Jr., Dennis, A.R., Valacich, J.S., Vogel, D.R., & George, J.F. (1991a). Electronic meeting systems to support group work: theory and practice at Arizona. *Communications of the ACM,* 34(7) (pp. 40-61).

O'Leary, D.E. (1998). Enterprise knowledge management. *IEEE Computer,* 31(3) (pp. 54-61).

Orwig, R.E., Chen, H., & Nunamaker, J.F. (1997). A graphical, self-organizing approach to classifying electronic meeting output. *Journal of the American Society for Information Science,* 48(2) (pp. 157-170).

Pedersen, J.O., Schmeltz, G. (1993). A browser for bibliographic information retrieval, based on an application of lattice theory. In *Proceedings of the 16th Annual International ACM/SIGIR Conference,* (pp. 270-279), Pittsburgh, PA.

Pinkerton, B. (1994). Finding what people want: Experiences with the WebCrawler. *Proceedings of the Second International World Wide Web Conference '94*, October 17-20, 1994, Chicago, IL.

Pirolli, P. & Card, S. (1995). Information foraging in information access environments. Proceedings of the *Conference on Human Factors in Computing Systems.*

Pirolli, P., Schank, P., Hearst, M.A., & Diehl, C. (1996). Scatter/Gather Browsing Communicates the Topics Structure of a Very Large Text Collection, *Proc. ACM CHI96 Conference*, April 13-18.

Plaisant, C., Bruns, T., Shneiderman, B., & Doan, K. (1997). Query previews in networked information systems: the case of EOSDIS. In *Proceedings of ACM CHI 97 Conference on Human Factors in Computing Systems* (pp. 202-203).

Rao, R., Pedersen, J.O., Hearst, M.A., & Mackinlay, J.D. (1995). Rich Interaction in the Digital Library. *Communications of the ACM*, 38(4) (pp. 29-39).

Rasmussen, E. (1992). *Clustering Algorithms.* In W.B. Frakes & R. Baeza-Yates (Eds.), Information Retrieval, Data Structures & Algorithms. Englewood Cliffs, New Jersey, Prentice Hall (pp. 419-442).

Ritter, H., & Kohonen, T. Self-organizing semantic maps. Biological Cybernetics, 61:241-254, 1989.

Roussinov, D. (1999). Internet Search Using Adaptive Visualization, in proceedings of *ACM SIGCHI 1999 Conference on Human Factors in Computing Systems* (pp. 69-70), May 15-20, Pittsburgh, PA.

Roussinov, D., & Chen, H. (1998). A Scalable Self-organizing Map Algorithm for Textual Classification: A Neural Network Approach to Thesaurus Generation, in *Communication and Cognition -- Artificial Intelligence*, 15 (1-2), 1998 (pp. 81-112).

Roussinov, D., & Chen, H. (1998a), Improving Similarity Computation for Clustering Documents: an Iterative Similarity Approach. AI Lab. Technical Report.

Roussinov, D., & Chen, H. (1999). Document Clustering For Electronic Meetings: An Experimental Comparison Of Two Techniques, *Decision Support Systems*, to appear.

Roussinov, D., & Ramsey, M. (1998). Information forage through adaptive visualization, in proceedings of *The Third ACM Conference on Digital Libraries* (pp. 303-304), June 23-26, 1998, Pittsburgh, PA.

Sahami, M., Yusufali, S., & Baldonado, Q.W. (1998). SONIA: A Service for Organizing Networked Information Autonomously. *Proceeding of the 3nd ACM International Conference on Digital Libraries* (pp. 237-246). Pittsburgh.

Salton, G. (1989). Automatic text processing: the transformation, analysis, and retrieval of information by computer. Reading, MA: Addison-Wesley.

Salton, G., & McGill, M.J. (1983). *Introduction to Modern Information Retrieval*. New York. McGraw-Hill.

Schatz, B., Mischo, B., Cole, T., Hardin, J., Bishop, A., & Chen, H. (1996). Federating Diverse Collections of Scientific Literature. IEEE Computer. 29(5) (pp. 28-36).

Schatz, B.R., and Chen, H. (1996). Building large-scale digital libraries. IEEE COMPUTER, 29(5):22-27, May 1996.

Shneiderman, B. (1996). The eyes have it: A task by data type taxonomy. In *Proceedings of Visual Languages*, Boulder, CO.

Shneiderman, B., & Kandogan, E. (1997). Elastic windows: Evaluation of multi-window operations. In *Proceedings of ACM CHI 97 Conference on Human Factors in Computing Systems* (pp. 250-257).

Shneiderman, B., Byrd, D., & Croft, W.B. (1998). Sorting out searching: A user-interface framework for text searches. *Communications of the ACM*, 41 (pp. 95-98).

Simon, A. (1991). Artificial Intelligence: where has it been, and where is it going? *IEEE transaction on knowledge and data engineering*, 3(2).

Tufte, E. (1990). Envisioning information. Chelshire, CT: Graphics Press.

Voorhees, E.M. (1985). The clustering hypothesis revisited, In Proceedings of *ACM Conference on Research and Development in Information Retrieval* (pp. 188-196).

Wagner, R.L. and Engelmann, E. (1997). Building and managing the corporate intranet. McGraw-Hill, New York.

Wang Baldonado, M.Q., & Winograd, T. (1997). SenseMaker: An information-exploration interface supporting the contextual evolution of a user's interests. *Proceedings of the ACM/SIGCHI Conference on Human Factors in Computing Systems* (pp. 11-18). Atlanta, GA.

Willett, P. (1988). Recent trends in hierarchical document clustering: A critical review. *Information Processing and Management*, 24(5) (pp. 577-597).

Young, D., & Shneiderman, B. (1993). A graphical filter/flow model for Boolean queries: An implementation and experiment. *Journal of the American Society for Information Science*, 44 (pp. 327-339).

Zamir, O., Etzioni, O., Madani, O., & Karp, R.M. (1997). Fast and intuitive clustering of Web documents. *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining* (pp. 287-290).