8-5-2006

Scribe: A Clustering Approach To Semantic Information Retrieval

Joseph R. Langley

SCRIBE: A CLUSTERING APPROACH TO SEMANTIC INFORMATION RETRIEVAL

By

Joseph Russell Langley

A Thesis
Submitted to the Faculty of
Mississippi State University
in Partial Fulfillment of the Requirements
for the Degree of Master of Science
in Computer Science
in the Department of Computer Science and Engineering

Mississippi State, Mississippi

August 2006

SCRIBE: A CLUSTERING APPROACH TO SEMANTIC INFORMATION RETRIEVAL

By

Joseph Russell Langley

Approved:

Susan M. Bridges Professor of Computer Science and Engineering (Major Professor and Co-Director of Thesis)	Julia E. Hodges Head and Professor of Computer Science and Engineering (Committee Member)
J. Edward Swan II Associate Professor of Computer Science and Engineering (Co-Director of Thesis)	Edward Allen Associate Professor of Computer Science and Engineering Graduate Coordinator

Kirk H. Schulz
Dean of the Bagley College of Engineering

Name: Joseph Russell Langley

Date of Degree: August 5, 2006

Institution: Mississippi State University

Major Field: Computer Science

Major Professor: Dr. Susan M. Bridges

Title of Study: SCRIBE: A CLUSTERING APPROACH TO SEMANTIC INFORMATION RETRIEVAL

Pages in Study: 92

Candidate for Degree of Master of Science

Information retrieval is the process of fulfilling a user's need for information by locating items in a data collection that are similar to a complex query that is often posed in natural language. Latent Semantic Indexing (LSI) was the predominant technique employed at the National Institute of Standards and Technology's Text Retrieval Conference for many years until limitations of its scalability to large data sets were discovered. This thesis describes SCRIBE, a modification of LSI with improved scalability. SCRIBE clusters its semantic index into discrete volumes described by high-dimensional extensions to computer graphics data structures. SCRIBE's clustering strategy limits the number of items that must be searched and provides for sub-linear time complexity in the number of documents. Experimental results with a large, natural language document collection demonstrate that SCRIBE achieves retrieval accuracy similar to LSI but requires 1/10 the time.

DEDICATION

I dedicate this research to my parents, Randy and Diane, and to my sister, Rachel.

ACKNOWLEDGMENTS

I wish to express my gratitude: to Dr. Susan Bridges, my major professor, for her guidance and advice; to my graduate committee members, Dr. Julia Hodges and Dr. Ed Swan; and to Dr. Stephanie Doane for her encouragement and support. My thanks also go to Dr. T. J. Jankun-Kelly for his patience during countless spontaneous office visits and for pointing out some excellent reading material on signed distance fields.

I also wish to thank my colleagues in the Department of Psychology for their tutelage in the mysteries of the ANOVA: Dr. Deborah Eakin, Dr. Carrick Williams, Teena Garrison, and Mark Thomas.

Finally, I extend my thanks to Dr. Len Miller of the Department of Mathematics and Statistics for donating his vector analysis textbook to my work.

TABLE OF CONTENTS

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

1.1 Background

Information retrieval is loosely defined as the task of informing users of the existence and location of information related to their requests. Information retrieval (IR) systems do not change or add to the information contained in the items they return to the user. Such systems are not necessarily intelligent – in the sense of intelligent agents – though many examples are [44]. The goal of this thesis is to improve an information retrieval technique – Latent Semantic Indexing (LSI) – by combining its elegant approach to conceptual clustering with a simplified representation of high-dimensional "semantic spaces" adapted from efficient data structures previously developed for use in computer graphics. In order to accomplish this goal, we have developed data structures and algorithms, collectively named SCRIBE (Semantic Cluster Retrieval Index Basic Elements), to efficiently search a semantic index while maintaining retrieval performance comparable to legacy systems. This approach restores the competitiveness of LSI-like retrieval solutions with other contemporary IR systems. Experimental results with the OHSUMED document collection [30] demonstrate that SCRIBE performs as well as LSI but takes less than $1/10^{th}$ as long to answer users' queries.

1.1.1 Information Retrieval

Information retrieval is often defined in terms of its contrast with the related problem of data retrieval. Data retrieval is generally tackled as a pattern-matching task in which a highly structured query in a specialized language describes attributes of the desired subset of a data collection. Data retrieval (DR) tends to be computer-centric and well suited to serving the needs of automatic algorithms. Relational databases provide an example of the characteristics of data retrieval systems. Information retrieval, on the other hand, addresses situations to which deterministic solutions prove unsatisfactory and in which the human factors involved preclude the simplifying assumptions of DR algorithms. Hallmarks of situations that call for IR include the need for natural language queries, the preference toward partial matching of query attributes, recall of relevant[1] information from partially matching branches of (what would be) the recognition decision tree, and the need to seek information among very large data sets. Information retrieval is closely related to – and often considered to be synonymous with – document retrieval, in which natural language documents are parsed and indexed in order to serve natural language queries. Document retrieval is a special case of information retrieval; IR addresses a wider range of problems [3].

Information retrieval (IR) is most often applied to data sources in which the information is encoded in a human-understandable format, but not one easily parsed by computers – e.g., conversational speech or college essays. Google is one of the best-

[1] Relevance is a metric defined by the application domain. One may consider that in response to the query, "cancer cells," medical journal articles are likely more relevant than movie reviews.

known information retrieval applications; virtually every Internet aficionado has used Google's document searching tools that index and serve links to billions of web pages. Other common applications include retrieval from bioinformatics sequence repositories, specialized document collections with abnormal data distributions, imagery databases, spoken word corpora, and cross-language document collections. A common thread among these problems is the difficulty of instructing a computer to understand the data retrieved by the IR process. Rather than understanding the data they search, IR systems compare queries to a statistical model and use knowledge gleaned from the comparison to guide a search through the document collection. A good IR system contains a data model that produces results consistent with human intuition – though the model itself is rarely transparent to intuition [3].

1.1.2 Latent Semantic Indexing

IR techniques excel at distilling relationships among raw data units into weights, connections, or rules for recognizing similarity – they simplify the features of complex data into automatically learnable units. Depending upon the technique we choose to apply, we might think of the learnable units in terms of simpler metaphors – support vectors, hidden states, and connection weights – or we might choose a data-driven approach that requires little or no interpretation. One such data-driven approach, Latent Semantic Indexing (LSI), was developed by Dumais, Deerwester, Landauer, Foltz, et al. at Bellcore Labs around 1988 [17]. LSI, an IR technique commonly applied to the document retrieval problem, requires minimal parsing of documents and constructs powerful classification models from very simple data structures. LSI was introduced to

the IR community at the First Text Retrieval Conference (TREC) hosted by the National Institute for Standards and Technology (NIST) in 1993 [13, 17]. TREC is a competitive conference offering several tracks or data sets geared to exercise IR systems' ability to solve problems in domains of interest to the government customer community. Tracks include cross-language retrieval, question answering, and large document collection retrieval. LSI routinely exhibited retrieval accuracy on par with support vector machines, its highest-scoring contemporary at TREC [13, 14, 16]. LSI so impressed the community that more than half of the entries in the second TREC (1994) were based upon LSI-like techniques [14].

LSI's strong performance at TREC stems from its ability to infer knowledge of synonymous and polysemous words from their occurrence in similar contexts. Thus, LSI is able to recognize and retrieve documents conceptually similar to users' queries even when the documents and queries are not lexicographically similar. Its strength at induction, however, is not matched by its search strategy for responding to queries.

To understand the LSI approach, consider Google's task – locating documents that are similar to a short, user-generated query string. Table 1.1 contains a collection of documents; for this example, the collection consists only of titles. The documents span a range of topics and share common words that differ in meaning according to their context. The common words – those that appear in more than one document – are italicized for emphasis. Each document is converted to an ordered, column vector with common word counts as the elements. (For non-trivial collections, the raw occurrence counts are replaced with a weighted frequency metric.) The column vectors are

juxtaposed to create a "term-by-document" or "term-document" matrix from which the final index is computed.

Table 1.1 Titles of technical memos as represented in a co-occurrence matrix. From: Deerwester, 1990 [11].

Technical Memo Example

Titles:
c1: *Human* machine *interface* for Lab ABC *computer* applications
c2: A *survey* of *user* opinion of *computer system response time*
c3: The *EPS user interface* management *system*
c4: *System* and *human system* engineering testing of *EPS*
c5: Relation of *user*-perceived *response time* to error measurement
m1: The generation of random, binary, unordered *trees*
m2: The intersection *graph* of paths in *trees*
m3: *Graph minors* IV: Widths of *trees* and well-quasi-ordering
m4: *Graph minors*: A *survey*

Terms	**Documents**								
	c1	c2	c3	c4	c5	m1	m2	m3	m4
Human	1	0	0	1	0	0	0	0	0
interface	1	0	1	0	0	0	0	0	0
computer	1	1	0	0	0	0	0	0	0
User	0	1	1	0	1	0	0	0	0
System	0	1	1	2	0	0	0	0	0
response	0	1	0	0	1	0	0	0	0
Time	0	1	0	0	1	0	0	0	0
EPS	0	0	1	1	0	0	0	0	0
Survey	0	1	0	0	0	0	0	0	1
Trees	0	0	0	0	0	1	1	1	0
Graph	0	0	0	0	0	0	1	1	1
Minors	0	0	0	0	0	0	0	1	1

Computation of a latent semantic index proceeds along the assumption that the meaning of a given document is encoded in a "semantic space" – a high-dimensional space in which the axes represent components of conceptual meta-language that are independent of the specific language in which a document is written. Natural languages form analogous "lexical spaces" which contain an axis for every word in the language. In general, the semantic space derived from a document collection has orders of magnitude

fewer dimensions than the corresponding lexical space. Lexical spaces contain so many more dimensions because of synonymy, the occurrence of many words with the same meaning. Noise is also introduced to lexical spaces through polysemy, the presence of words with many meanings. In short, synonymy and polysemy increase the number of ways to describe identical concepts, but the meaning of a given concept is unique. We induce semantic knowledge into the index by using principle component analysis to find a reduced-rank approximation of the term-document matrix, a sample of the lexical space of the document collection. When the approximation's rank is close to the dimensionality of the semantic space, the index will contain term-document vectors that cluster according to their conceptual content. Table 1.2 shows a rank-2 semantic index computed from the term-document matrix composed in Table 1.1. In this example, dimension reduction was accomplished via manipulation of the singular value decomposition (SVD) of the term-document matrix. We select a set of components associated with the greatest N singular values in the decomposition where N is the target dimensionality of the index, and re-compose those singular vectors into the final index. Figure 1.1 illustrates the SVD and composition of the semantic index.

Table 1.2 Rank-2 approximation of term-document co-occurrence matrix. From: Deerwester, 1990 [11].

Terms	Documents								
	c1	c2	c3	c4	c5	m1	m2	m3	m4
human	0.16	0.40	0.38	0.47	0.18	-0.05	-0.12	-0.16	-0.09
interface	0.14	0.37	0.33	0.40	0.16	-0.03	-0.07	-0.10	-0.04
computer	0.15	0.51	0.36	0.41	0.24	0.02	0.06	0.09	0.12
user	0.26	0.84	0.61	0.70	0.39	0.03	0.08	0.12	0.19
system	0.45	1.23	1.05	1.27	0.56	-0.7	-0.15	-0.21	-0.05
response	0.16	0.58	0.38	0.42	0.28	0.06	0.13	0.19	0.22
time	0.16	0.58	0.38	0.42	0.28	0.06	0.13	0.19	0.22
EPS	0.22	0.55	0.51	0.63	0.24	-0.07	-0.14	-0.20	-0.11
survey	0.10	0.53	0.23	0.21	0.27	0.14	0.31	0.44	0.42
trees	-0.06	0.23	-0.14	-0.27	0.14	0.24	0.55	0.77	0.66
graph	-0.06	0.34	-0.15	-0.30	0.20	0.31	0.69	0.98	0.85
minors	-0.04	0.25	-0.10	-0.21	0.15	0.22	0.50	0.71	0.62

Query processing projects the vector form of a query string into semantic space for comparison with the documents' column vectors. Each query is converted to a term vector according to the (weighted) occurrence counts of indexed terms in the query string. The query vector is multiplied by the term vectors of the SVD (matrix U_k in Figure 1.1) and the diagonal matrix of singular values (Σ_k) to project it into the semantic space. The resulting coordinates are compared with each document column vector in V_k (as scaled by the matrix Σ_k) to produce a similarity metric. The result returned for a query is the list of documents in the index as ordered by the similarity metric (or, more often, an ordered subset of the most similar documents).

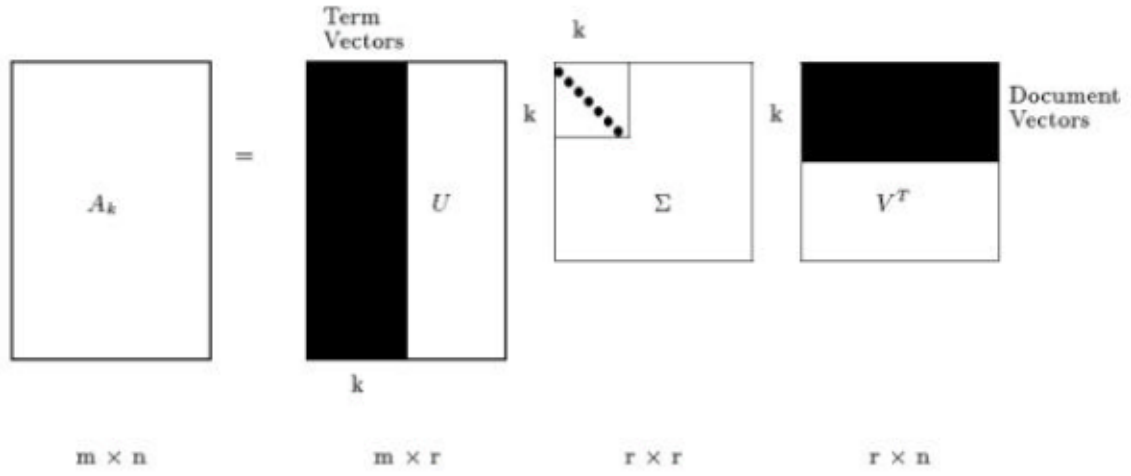

Figure 1.1 Visual representation of the Singular Value Decomposition of a rank-r matrix A. Dimension reduction involves selecting the k greatest singular values from Σ and the k corresponding column vectors in U and V. Re-composition of U_k, Σ_k, and V_k produces A_k, the rank-k approximation of A. From: Dumais, 1995 [15].

1.1.3 Weaknesses of LSI

LSI fell out of vogue because the canonical search procedure involves comparing each of N documents in the index to the query string followed by a sort of the result list. Each comparison includes a similarity metric calculation of linear order in M, the number of indexed terms. Thus, the search procedure's asymptotic time complexity is O(MNlogN) in the best case.

Semantic indices are also enormous; not as large as the original document collection, but not as compact as the models produced and employed by other IR algorithms, either. The minimal storage charge for a semantic index is of order O(k*(M+N)) where k is the dimensionality of the index (the number of singular values kept after dimension reduction).

Adding new data to a semantic index incurs a cost either in reprocessing the document collection into a new index or in warping the axes of the existing index. This makes online learning of models very difficult. Use of certain other matrix decompositions shows promise in reducing the warping [2, 34].

1.2 Motivation

In the decade since LSI's introduction, SVM and other IR systems have solved the synonymy and polysemy problems in their idiomatic ways and with much faster searches. LSI has been boosted a few times by the addition of highly parallelized search algorithms, space partitioning searches, and index-shrinking strategies. These additions and modifications have kept LSI commercially viable, but haven't spurred further innovation. The simplicity and elegance of the LSI approach to information retrieval mark it as a perfect candidate for early analysis of any large data set in which the connections between concepts are hypothesized but not fully understood. LSI is not as well suited to implementation in COTS hardware as other IR applications. LSI is also patented – unlike many other IR systems; the cost of licensing it has decreased its popularity as a prototyping tool and its license terms have restricted the third party innovation that bolstered its competitors at TREC.

The approach of this thesis is to construct a process that operates upon an LSI-like semantic index to superimpose smooth, closed surfaces over the document clusters in a semantic space. Once an orthonormal basis for the semantic space has been extracted or inferred, we only need to store the control points of the superimposed surfaces in order to completely describe the conceptual clusters. The surface representations facilitate search

algorithms with improved time complexity over existing LSI searches while maintaining the same accuracy. This approach also produces conceptual clusters that can be incorporated into other machine learning systems.

1.3 Hypothesis

Latent semantic indexing is an effective information retrieval technique, but search efficiency scales poorly and storage needs are unacceptable for large data collections. Techniques derived from machine learning and computer graphics can be applied to semantic indices to generate data structures that support more efficient search while maintaining comparable retrieval accuracy.

1.4 Contributions

Current information retrieval methods based on semantic indexing are effective but inefficient with large data collections. Contributions of this thesis are the development of:

- Clustering parameters suitable for partitioning semantic spaces into conceptually similar groups of documents;
- Data structures adapted from 3D computer graphics suitable for compact storage and manipulation of document cluster boundary surfaces;
- Time-efficient search algorithms based upon ray tracing methods.

The new data structures and methods also allow incremental updating of data collection indices with new documents, a procedure which is not currently supported by LSI.

1.5 Applications

By reducing the search and storage complexity of semantic indexes, we have provided mechanisms that can potentially be used for applying LSI-like techniques to new commercial application domains such as virus scanning, network intrusion detection, and spam filtering. These problems can all potentially benefit from integration of the knowledge gained from semantic indexing with other machine learning algorithms. By providing concept primitives derived from the semantic index, we support learning of better models for these applications and more.

1.6 Organization

The remainder of this document is organized as follows:

Chapter II outlines the body of literature related to this thesis. Chapter III outlines the research approach that was followed and describes design parameters for the SCRIBE information retrieval pipeline. Chapter IV discusses the design choices that were made during implementation of the SCRIBE system. Chapter V presents experimental results demonstrating statistical identity between the retrieval performances of the SCRIBE and LSI systems and dramatic improvement in SCRIBE's running time compared to LSI. Chapter VI discusses the conclusions of this thesis, summarizes its contributions to the field of research, and suggests some topics of future inquiry.

CHAPTER II

LITERATURE REVIEW

The goal of this thesis research is to improve the scalability of latent semantic indexing (LSI) by applying clustering methods from machine learning and search methods derived from computer graphics. LSI has been used both for information retrieval (IR) and as part of tools for textual analysis [19, 20, 23]. The representations and algorithms developed in this research improve the performance of LSI for information retrieval and, by extension, textual analysis. Section 2.1 provides a brief review of IR methods with an emphasis on vector model methods. Clustering methods will be used to extract characteristic features from LSI and are reviewed in Section 2.2. Methods from computer graphics have been used to develop compact representations for clusters. These methods are discussed in Section 2.3. Section 2.4 presents a review of ray tracing methods that are used to represent queries of the new, geometric cluster representations.

2.1 Information Retrieval

Information retrieval (IR) is the process of fulfilling a "user information need" [3]. The process subsumes simple data retrieval and additionally includes filtering and ranking steps that aid users' comprehension. Baeza-Yeats and Ribeiro-Neto [3] describe the three "classic models" of information retrieval: the Boolean, probabilistic, and vector models.

All three models consider documents as bags of words characterized by index terms and assume that index terms are mutually independent and orthogonal. Each model is distinguished by a particular representation of documents and user queries.

The Boolean model represents documents as binary strings in which each bit indicates the presence (1) or absence (0) of an index term. Queries are posed as conjunctive forms that specify constraints on the set of documents that can potentially fill the user information need. Retrieval in the Boolean model is a binary decision – if a document fully satisfies the query, it is retrieved; otherwise the document is discarded. No partial matching is allowed. All retrieved documents satisfy the query equally, so there is no ranking of the items in the returned set [3]. Because it lacks the ability to retrieve partial matches and rank documents, the Boolean model is little better than data retrieval at fulfilling users' information needs. Thus, it has little value to IR research, though it has achieved some commercial popularity.

Probabilistic IR approximates an "ideal answer set" [3] by estimating the likelihood of relevance to a user query for each document in the collection. This model represents both documents and queries as binary strings like those described in the Boolean model. During retrieval, a probabilistic model of relevance is learned by iteratively refining estimates of the probability that each index term occurs in at least one relevant document. The probabilities are refined by applying the estimation rules to the set of documents retrieved by the previous set of probabilities. (Initially, all probabilities are set to 0.50.) After several iterations, the model is expected to converge on an approximation of the ideal answer set. Documents in the final answer set are ranked in

order of their probability of relevance. Probabilistic IR can retrieve documents that only partially match a user query. As in the Boolean model, index term weights are binary. Document ranks are based on the sums of term weights, so ranks in the probabilistic model do not reflect the rate of a term's occurrence in a given document; all other factors being equal, a document in which term *t* occurs fifty times is given the same rank as one in which *t* occurs only once [3]. In practice, this can mean that technical documents on a subject may be ranked equally with tabloid articles on the same topics. Baeza-Yeats and Ribeiro-Neto [3] review the most popular probabilistic models: Bayesian networks, inference networks, and belief networks.

The vector model approaches IR as a clustering problem. Documents are considered as points in a high-dimensional lexical space in which the frequency of each index term is plotted along one of many mutually orthogonal axes. Documents and user queries are represented as vectors of index term frequencies or weights. In the vector model, term weights derived from the rate of occurrence of an index term in a document are substituted for raw term frequencies. Thus, each document vector uniquely identifies a point in lexical space. Documents are ranked according to their similarity to a user query. The vector model's concept of similarity is the degree to which two vectors point in the same direction (e.g., are members of the same cluster). This concept is most often captured in metric forms utilizing the dot product or cosine functions. The vector model's formulation of similarity allows partial matching of user queries. Because index term weights come from the domain $[0, \infty)$, documents with high correlation to specific terms in a user query (i.e., have high intra-cluster correlation) tend to be ranked above

documents with low correlation to all terms in the user query (i.e. have high inter-cluster correlation) [3].

2.1.1 Vector Model Techniques for LSI

Latent Semantic Indexing (LSI) processes a data collection to generate a model of the information contained in a data collection's addressable parts [11]. The information model is used to inform online recognition of the information represented by new data samples and queries, and facilitates the retrieval of similar documents from within the indexed collection. The data and information models are constructed in roughly the same data structures - two-dimensional, rectangular matrices. Rows and columns of the matrices represent two levels of granularity at which the data is analyzed to construct the information model: columns represent the addressable or retrievable items in the data collection such as documents, images, or sound files; each row represents an atomic data element parsed from the addressable items such as a word, pixel, or pitch. Because the number of rows is usually much greater than the number of columns the data matrix – called a co-occurrence matrix – tends to be sparse. As the induction procedure resolves polysemy and synonymy relationships the initial atom weights are spread and shared among groups of related atoms [4]. Because of the spread of atom weights, the matrix representing the information model is always dense [17].

Davis and Foltz [10] discuss the problem of learning recognition models from example data by compacting the data representation using Minimum Description Length – the least number of bits in which a message can be encoded – as a heuristic to guide an automatic search for an efficient encoding. In particular, they treat the problem of data

compression. For example, in the language $L = \{A, B, C, D\}^*$ certain character combinations might occur so frequently that the number of transmitted characters can be reduced by including special characters to represent those groups in the data model. By analyzing the growth of parsimony networks – hierarchical networks representing decisions to merge character co-occurrences into special characters - Davis and Foltz found a method for learning data recognition models that account for polysemous alphabets by merging characters whenever the merge minimizes the model's total description length. They found that merging characters that encoded similar events in the language tended to reduce the description length of the data model and improve recognition accuracy.

In LSI, Dumais, Furnas, Landauer, et al. [11] applied an encoding compaction technique to a vector space model of natural language in which each word in the language is mapped to a unique axis on a high-dimensional orthonormal basis. Example documents are encoded as word-count vectors in the language space and the vectors are juxtaposed as columns in a sparse matrix. Encoding compaction was interpreted as an analogous process to projection of the language space – or lexical space – into a lower-dimensional "semantic space". This projection embodies an underlying assumption that a hidden semantic grammar guides the choice of words in natural language documents. By projecting the document vectors into a space of approximately the same dimensionality as the semantic space, it was expected that a model of the information content of the example documents would be induced. Experiments at TREC [13, 14, 16] and in several user studies [19, 21, 22, 37] indicate that the dimension reduction method does capture an

information model with characteristics similar to the models described by Davis and Foltz [10]. The projection technique described by Dumais, Furnas, Landauer, et al. [11] reduces description length with respect to the number of axes in the lexical and semantic spaces, but tends to increase the space needed to store the induced model by mapping the sparse lexical matrix onto a dense semantic matrix. The gain in recognition accuracy comes at premium in storage complexity. In analogy to Davis and Foltz's [10] parsimony networks, orthonormal axes replace atoms of a language and atom merging is replaced by the projection of related axes onto a new, composite axis [4, 10, 17].

The choice of which dimensions to merge is decided by a form of principle component analysis derived from a linear algebraic technique called Singular Value Decomposition (SVD). The details of the SVD are discussed in Chapter 1. In this chapter we will highlight the motivation for applying matrix decomposition to knowledge induction. SVD divides the co-occurrence matrix into three components: the left singular matrix represents a transform that describes an orthonormal basis; the singular vector – a diagonal matrix – contains the PCA results and describes the information and noise content of each axis; the right singular matrix represents the coordinates of the addressable items with respect to the orthonormal basis. When multiplied together, these three components exactly reproduce the original co-occurrence matrix. Davis, Foltz, and Dumais [10, 11, 17] demonstrated that reducing the dimensionality of a model induces knowledge about the underlying (or "latent") information structure with the greatest information gain occurring when the target number of dimensions is set very close to that of semantic space. This semantic space is a conceptualization of the vector space

corresponding to a hypothesized semantic meta-language that directs word choices independently of the human language in which a document is composed. The dimensionality of semantic space cannot be measured directly; rather, the dimensionality of an information model is said to be close to that of the semantic space whenever the model's retrieval metrics are within experimental or production tolerances. As applied to the SVD, each singular value and its associated pair of singular vectors represent a component of the transform that maps an atom-count vector into lexical space [4, 17]. Each component transform encodes information about the atom-count vectors' positional relationships along one axis of the orthonormal basis of lexical space. Dumais, Furnas, Landauer, et al [17] found they could choose dimensions to merge based upon the PCA results embodied by the singular values. Their method, LSI, approximates the semantic space of a data collection by retaining from the full SVD only the transforms associated with the k greatest singular values. In effect, the information contained in the discarded transforms is merged into those that remain – an analog to the parsimony network merges described by Davis and Foltz [10].

A desirable side effect of dimension reduction is the formation of clusters within the semantic index. Atom-count vectors are scaled and rotated during the SVD process such that conceptually similar data are moved close together in the approximated semantic space [17]. Davis and Foltz [10] describe the information structure of semantic indices in analogy to artificial neural networks. They interpret the dimensions of document vectors as edge weights leading from an input array into an array of neurons in which each document vector maps to a unique neuron that recognizes the document from

which its edge weights were derived. This interpretation is also similar to Kohonen's [33] self-organizing map, in which a 2D array of neurons is connected to an N-D input space. The SVD moves the vectors themselves into closer proximity whereas Kohonen's learning rule adjusts the weights of fixed nodes to detect the presence of stimuli in discrete regions of the input space. SVD and self-organizing maps will be compared further in the coming sections.

2.1.2 Queries in IR Systems

We have seen how LSI, as an example of vector space models, represents information and learned knowledge. To begin our discussion of information retrieval, let us examine the vector representation of queries. Queries may take one of two general forms: (1) short, keyword-rich descriptions of the desired information, or (2) example items that are similar to the desired information (for example an article on the topic of interest might be used to query the Lexus Nexus database of journal articles). LSI processes both query forms identically. First, an atom-count vector is constructed by counting the atoms (e.g., words) in the query. The atom-count vector is mapped to a template derived from the rows of the retrieval index – that is from the indexed terms in the semantic space. Counts for any atoms that match indexed terms are copied into the template leaving behind any non-indexed atoms. The filled template is scaled and rotated into semantic space by multiplication with the basis transform computed during SVD [4, 17]. After the basis transformation, the query vector represents the coordinates of a point along a new "pseudo-document" vector in semantic space [11].

For each query, LSI ranks every document vector (addressable item) in its index with respect to the query's pseudo-document vector though only the few highest-ranked datum are actually included in the response to the user [11]. The ranking procedure is roughly equivalent to K-Nearest Neighbor learning. The distance between the query's pseudo-document and each indexed item is computed. A rank-ordered list of the indexed items is then constructed and the top few items are returned in the query response [11, 15]. Note that the distance computed between the pseudo-document and the indexed items is usually not the Euclidean distance between these points; most often, the metric is either the dot product of the two vectors or the cosine of their included angle [15, 36]. Dumais first noted the scalability concerns surrounding the exhaustive search of the index in the SIAM Review in 1995 [15]. Chen et al. [9] discuss implementation strategies to overcome some scalability hurdles in a 2001 technical report.

2.2 Cluster Extraction

Hand et al. [28] define clustering as the process of "decomposing … a … data set into groups so that the points in one group are similar to each other and are as different as possible from the points in other groups." This thesis research applies clustering methods from machine learning to semantic indices in order to extract information about the distribution of data points in the index and the clusters they are expected to form. Hand et al. divide clustering methods into three categories: hierarchical, partition-based, and probabilistic. The three categories are defined based on the scope of the scoring function (local or global), and on the determinism of the search method used to pick candidate cluster members (see Figure 2.1).

Hierarchical clustering algorithms characteristically include a local scoring function and search method that is tightly correlated with a specific, expected cluster model [28]. Peter Willett [48] presents a comprehensive review of hierarchical clustering strategies used for information retrieval through 1988. Koller and Sahami [35] developed an on-line learning technique for extending hierarchical cluster graphs. This on-line technique superimposes over the cluster hierarchy a decision network in which each decision depends on a small subset of the characteristic features of the underlying cluster. Broder et al. [8] published a comprehensive, syntactic clustering of the entire World Wide Web in 1997. Broder's hierarchical method includes some novel similarity metrics, but is most impressive because of its scalability; Broder's test data included 30 million hypertext documents which were clustered and indexed for searching in about 10.5 days of processing time. Karypis, Han, and Kumar [32] presented Chameleon, a hierarchical clustering method that uses a hybrid, adaptive cluster model, in 1999. Chameleon operates on a k-Nearest Neighbor graph derived from the raw data. The cluster model considers metrics based on min-cut division of the vertices assigned to a cluster (local) and min-cut division of the links between clusters (global). The best candidate merge simultaneously maximizes both metrics; as a result, clusters of widely varying sizes and shapes can be discovered by Chameleon's single, adaptive model.

<table>
<tr><td colspan="2" rowspan="2"></td><td colspan="2">Search Strategy</td></tr>
<tr><td>Deterministic</td><td>Probabilistic</td></tr>
<tr><td rowspan="3">Scoring Scope</td><td>Local</td><td>Hierarchical</td><td rowspan="3">Fuzzy</td></tr>
<tr><td>Global</td><td rowspan="2">Partition-based</td></tr>
<tr><td>Hybrid</td></tr>
</table>

Figure 2.1 Categories of clustering algorithms

Partition-based clustering methods de-couple the cluster model from the scoring functions. Typically, they use global scoring functions – comparing a data point to all the possible clusters - and examine every data point during each iteration of cluster refinement. Thus, no specific cluster model is imposed by the clustering method, though the scoring function does attempt to enforce the previously mentioned similarity constraints [28]. Guha, Rastogi, and Shim [27] presented CURE, a partition-based clustering method for characterizing large databases, in 1998. CURE bears a strong likeness to the Chameleon system; CURE merges partial clusters formed by normalized, random sampling of the data points. After the initial partial clusters are merged, they act as scaffolds around which the whole database may be characterized and partitioned. Borodin, Ostrovsky, and Rabani [7] developed techniques for achieving near-quadratic time complexity in general when searching for optimal partitions of sparse clusters. (Sparse clusters are defined in Awerbuch and Peleg, 1990 cited in [7].) The use of Borodin et al.'s [7] technique applies the sparse cluster model to methods that are

otherwise independent of any cluster model. This loss of generality is usually acceptable compared to the gain in data throughput. McCallum, Nigam, and Ungar [38] proposed a further refinement to partition optimization in high-dimensional data sets – such as document collections – in 2000. McCallum et al. proposed a two-step method that first forms overlapping partitions across the entire data set and then efficiently refines cluster boundaries by examining only the data points that fall into overlapping regions. McCallum et al. report empirical findings of a 20x speed up over complete partition optimization with no significant change in IR metrics.

Probabilistic (or "fuzzy") clustering methods produce mixture models that predict a data point's membership in a given cluster with a known degree of error [28]. Fuzzy clusters are difficult to represent geometrically and are therefore of limited interest in the proposed research. However, unsupervised probabilistic clustering methods have been successful at predicting the number of clusters in large data sets – an accomplishment which is of great importance to this thesis research. Gath and Geva [26] developed an unsupervised learning algorithm based upon fuzzy k-means in 1989. Their method incrementally increased the number of clusters requested of the k-means algorithm until some performance metric function was maximized. Gath and Geva assume that the performance metric will be specialized for a given problem, so their method provides extensive flexibility for adaptation to almost any real-valued function. Slonim and Tishby [43] presented an information theoretic, fuzzy clustering algorithm for large, full-text databases in 2000. While their partitioning scheme is more sophisticated, Slonim

and Tishby also incrementally adjust the number of clusters over many iterations of their algorithm until some performance metric is maximized.

2.3 Geometric Surfaces

This thesis research adapts methods and data structures from computer graphics to compose compact representations for clusters. Three computer graphics approaches that were explored are implicit equations, non-uniform rational b-spline (NURBS) curves, and polygonal meshes. Each of these methods provides convenient data structures and algorithms for defining the boundary between points inside and outside of a given region of space. Shirley [42] describes implicit equations as real-valued functions that return zero whenever their arguments define a point on the "implicit surface." Shirley develops implicit equations for 3D planes and a small selection of 3D curves. He concludes by presenting techniques for representing surfaces in general by parameterized, implicit equations. Forrest [24] introduced Bezier curves to the computer graphics community in 1972. Bezier curves interpolate a continuous path from a set of control points according to a blending rule, a polynomial that defines the influence of each control point on the curve. Versprille [18, 45] describes non-uniform rational b-spline (NURBS) curves, the general framework of which Bezier curves are a special case. NURBS curves are piecewise functions in which each interval contains a Bezier blending function over the local control points. NURBS curves provide the advantage that changing a portion of a complex curve requires minimal re-computation of the blending polynomials. Hearn and Baker [29] describe polygonal meshes as the tessellation of a surface which approximates

the curvature with a set of polygonal patches. Shirley [42] describes data structures for efficiently representing meshes of triangular patches tangent to a curved surface.

2.4 Query Service and Ray-Surface Intersection

Query service in the vector space model of information retrieval is a mathematical analog to ray-surface intersection searches in computer graphics. This thesis research adapts optimizations of the ray-surface intersection search to improve the time complexity of LSI-like query service. Ray/scene intersection is the process in graphics of testing an arbitrary ray for intersection with all space-filling, geometric surfaces in the set of objects defined as a scene. In general, the ray/scene intersection is linear in the number of objects in the scene. Because scenes are usually sparse – having a low ratio of object volumes to total volume – space partitioning schemes frequently enable sub-linear complexity search and retrieval of objects that are likely to intersect with a given ray [42]. The clustering techniques described in the section 2.2 are spatial partitioning schemes. The document collections studied in this research are sufficiently large to demonstrate LSI's scale constraints, but are small enough that the document clusters produced by Chameleon, for example, sufficiently partition semantic space so that SCRIBE requires no further subdivision to maintain its search efficiency. When the techniques developed in this thesis – or, for that matter, any IR techniques – are applied to much larger document collections, it becomes necessary to generate a guide tree to direct the search algorithm toward regions of space that are likely to contain query matches. Two popular guide trees are BSP and k-d trees. Binary space partition trees (BSP-trees) are "binary trees for multidimensional points where successive levels are

split by arbitrary hyperplanes" [5]. That is, at each level the dimension containing the most information (as determined by an information gain metric) is chosen and a splitting hyperplane is constructed at the median value of that dimension among the objects in that level and section of the tree. At every decision point all dimensions are candidates for splitting. K-d trees [6] are a special case of BSP-trees in which the splitting dimension at each level is predetermined. For k-dimensional data the splitting dimension at level L is $L \bmod k + 1$. In practice, the depth of a k-d tree is roughly the same as a BSP-tree constructed on the same data (thus search complexity is asymptotically equal), but the construction time is reduced by a constant factor related to the time spent computing the information gain metric for the BSP-tree.

CHAPTER III

RESEARCH APPROACH

This thesis research considers information retrieval as part of the knowledge discovery and exploration framework in Figure 3.1. This framework provides mechanisms for fulfilling a user's information need by retrieving relevant documents through a natural language query service interface as required for information retrieval. In addition, the framework constructs the necessary data structures to permit visualization and exploration of the information space.

The research has followed the framework laid out in Figure 3.1 to test the hypothesis stated in Chapter I:

> *Latent semantic indexing is an effective information retrieval technique, but search efficiency scales poorly and storage needs are unacceptable for large data collections. Techniques derived from machine learning and computer graphics can be applied to semantic indices to generate data structures that support more efficient search with comparable retrieval accuracy.*

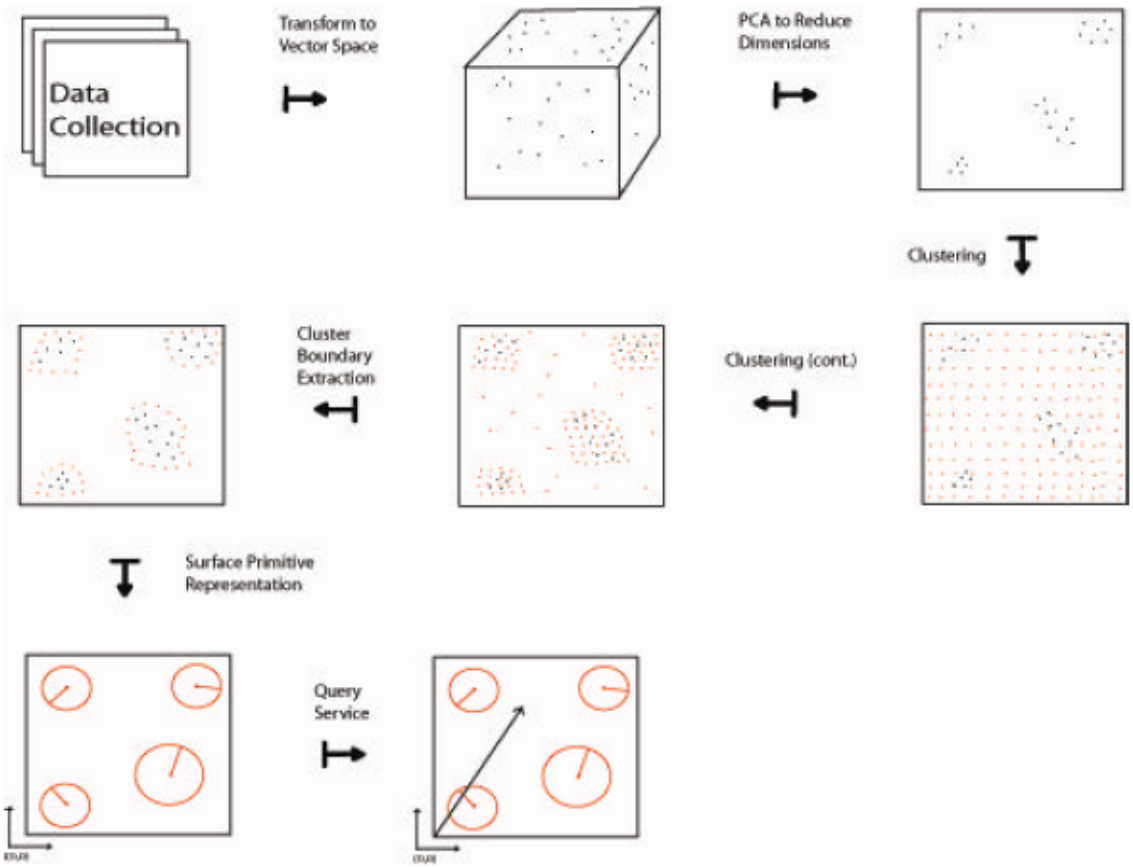

Figure 3.1 Proposed knowledge discovery and information retrieval framework

A prototype information retrieval (IR) pipeline has been constructed from the framework by selecting and adapting pre-existing algorithms for each task in the pipeline:

- Transformation to vector space,
- Dimensionality reduction,
- Clustering and boundary extraction,
- Surface representation,
- Query service.

The remainder of this chapter examines the tasks and the constraints each places upon candidate algorithms. This chapter concludes with a discussion of the metrics that were used to judge the success of the prototype.

3.2 Data Transformation to Vector Space

This task transforms a data collection from a relatively complex, free-form structure into the simplified representation required by the knowledge inference task that comes next in the pipeline. As discussed in the previous chapter, each of the three classic information retrieval models represents documents as strings of index term weights (either binary weights or real-values). The goal of this task, then, is the identification of index terms and the calculation of index term weights. This thesis research adopts a simple, common rule for choosing index terms: any term that appears in two or more documents is an index term. The particulars of calculating index term weights depend upon the representation model chosen. By focusing on the LSI approach, we have implicitly chosen to employ the vector space model of information retrieval. In addition, Dumais' study of term weighting techniques recommends the use of the local "log-frequency" and global "log-entropy" scheme, which we have also adopted [12]. The construction of these weight strings satisfies the representational requirements of the knowledge inference stage of any vector model-based IR pipeline.

3.3 Knowledge Inference by Dimension Reduction

This task involves learning the links between terms and information in the data collection. The vector model of IR frames this task as cluster analysis – documents that

contain similar information are expected, by virtue of the term weighting scheme, to occur relatively closer to each other than to dissimilar documents. This thesis research follows the example of previous implementations of the vector model that apply analytical algorithms to the initial document vectors to produce an information model. In particular, we apply principle component analysis and multi-dimensional scaling in the form suggested by Latent Semantic Indexing (LSI) [17]. The term-document matrix produced by the first stage of the pipeline is factored into its singular value decomposition (the PCA step) and approximated by its greatest eigenpairs (multi-dimensional scaling).

The semantic space produced by this stage of the pipeline can be searched as a document index itself – this is the strategy used by LSI. For the purposes of improving search efficiency and supporting visualization, however, this thesis research extends the IR pipeline to the discovery and discrete definition of document clusters.

3.4 Document Cluster Identification and Boundary Extraction

The index produced by the previous stage of the pipeline contains document clusters that represent the relationships among fragments of information in the collection. Those clusters, though, cannot be recovered from the index directly as might be desirable for visualization, classification, or other search optimizations as we have done in this research. This stage of the pipeline discovers the clusters formed during the inference stage and extracts their members into separate containers for further analysis.

The number of clusters formed during inference is unknown at this stage in the pipeline. Before proceeding to cluster identification, then, it is necessary to estimate the

number of clusters present in the semantic index. This is easily accomplished by a probabilistic clustering algorithm such as fuzzy k-means [26]. Probabilistic clustering was too time-consuming for the schedule of this thesis research. Preliminary trials with fuzzy k-means took days to complete one clustering solution out of the hundreds required for convergence. Instead, we developed a few simple criteria that describe acceptable clustering solutions. These criteria, described in the Appendix, were used to guide a manual search of the parameter space.

The choice of an appropriate cluster identification algorithm for this stage is constrained by the need to maintain the relative spatial separation between clusters and the need to identify clusters with both convex and concave geometries. As discussed in Chapter II, partition-based clustering algorithms are generally more able to adapt to complex cluster geometries than hierarchical or probabilistic algorithms. An exception, the hierarchical Chameleon system [32], is also a strong candidate. In this thesis research, we have adopted the Chameleon technique. First we compute a partition-based clustering solution over the semantic index. The small clusters of the first solution are then hierarchically merged.

3.5 Boundary Surface Representation by Graphics Primitives

This stage of the pipeline simplifies the representation of the extracted cluster volumes to improve storage and search complexity. The document vectors that make up the surface of each cluster volume are used as guidelines for fitting computer graphics primitives around each cluster. The choice of primitives is the users' prerogative; however, the complexity of the primitive chosen should agree with the complexity of the

clusters' geometries. For convex clusters, isometric primitives (e.g., spheres, cubes, etc.) may suffice. For concave clusters, primitives such as NURBS or polygonal meshes may be more appropriate. In our research, simplicity of the prototype pipeline has driven the choice of graphics primitives.

3.6 Query Service

At this stage of the pipeline, it is no longer necessary to search through all or most of the document vectors in the semantic index to find relevant matches to a user's query. With minor adaptation, the ray tracing method of intersection testing with the graphics primitives formed at stage 4 has been used to accomplish the task of locating relevant document clusters without actually examining any of the document vectors. We have adapted sub-linear ray tracing for use the IR pipeline. Adaptations include high-dimensional intersection tests, facilities for ranking document vectors in clusters intersected by a query vector, and cluster relevance rankings for non-intersected clusters.

3.7 Performance Metrics

The performance of the prototype pipeline has been compared with existing IR systems by the following metrics:

- Recall: the number of relevant documents retrieved;
- Precision: the ratio of relevant to non-relevant documents retrieved.
- F-measure: describes both recall and precision in a single value.

Experimental document collections available from the NIST Text Retrieval Conference (TREC) provide data sets, lists of relevant documents, and performance metrics supplied

by competitors. SCRIBE, an implementation of the prototype pipeline, was evaluated against a basic LSI implementation. Both systems were tested over W. Hersh's OHSUMED [30] document collection, queries, and relevance judgments from the TREC-9 filtering track.

CHAPTER IV

DESIGN AND IMPLEMENTATION

This chapter describes the design and implementation of the SCRIBE (Semantic Cluster Retrieval Index Basic Elements) system that was implemented as a proof of concept of the information retrieval pipeline discussed in Chapter I.

4.1 SCRIBE Design and Implementation

SCRIBE was constructed to implement the design shown in Figure 3.1 by adapting existing algorithms and building custom software for the six tasks below:

- Data transformation to vector space
- Knowledge inference by dimensionality reduction
- Clustering and boundary extraction
- Boundary surface representation
- Query processing.

4.1.1 Data Transformation to Vector Space

The design of this stage followed the examples of established vector model IR techniques. Documents were modeled as "bags of words" [3] that were converted to vector form by recording the raw occurrence counts for each term. Note that "terms" are differentiated from "words." A word is any non-null collection of alphanumeric characters. Terms are words with particular occurrence frequency characteristics. The

specific parameters that distinguish terms from words are user-selectable; in this work, terms were defined as words that occurred in more than two documents but less than 40% of documents in the collection. These parameter values were chosen to be typical of LSI systems as described by Foltz et al. in [10, 23, 36, 49].

All terms were compared to Salton's SMART stop list of high-frequency English words [41]. Terms appearing on this list occur so frequently in English natural language that modeling of them overwhelms the influence of information-bearing terms. Consequently, these terms are removed from the data stream early in the vector transformation process. Note that the limitation of term frequency to less than 40% of the document collection is intended to accomplish the same goal for domain-specific terms.

Additionally, the natural language document collection used in this work was stemmed – homogenized for variances in spelling resulting from use in different parts of speech, temporal sense, number, etc. – by applying the Porter Stemmer algorithm [39]. All documents, queries, and stop words were stemmed prior to any other vector space transformation step. The Porter Stemmer does not remove punctuation from a text stream; it was necessary to add some lightweight textual analysis procedures to the document vector construction program.

4.1.2 Dimension Reduction with SVD

Berry [4] published SVDPACKC, a reference implementation – in C – of several singular value decomposition (SVD) algorithms from the Linear Algebra Package (LAPACK) [1] algorithm collection. Rohde's simplified executable front-end [40] for the single-vector Lanczos SVD was used at this stage of the pipeline to perform

knowledge inference by dimension reduction. Rohde's SVD program permits customization of five parameters that affect the quality of the solution and performance of the algorithm. The first two parameters are the number of rows and columns in the term-document matrix (TDM). Control of these parameters is the most effective means of controlling the running time of the algorithm. However, the removal of rows that represent terms or columns that contain document vectors is not desirable because such actions cause information loss in the semantic index.

SVD rank is the third parameter. The rank of the decomposed matrix is analogous to the dimensionality of the term-document space. The SVD program can be told to end early when a certain number of the largest singular values have been discovered. This is advantageous, since this stage of the pipeline deals with dimension reduction. The rank of the SVD was therefore chosen as the target dimensionality of the semantic index. The specific value of this parameter was chosen by performance trials of the pipeline over a portion of the document collection reserved for training.

The final two parameters are intended to account for floating point rounding error, but can be used to affect adjustments in the granularity of the decomposition. The first of these parameters defines the range of values that will be interpreted as equal to zero. By default, this range is [-1E-30, 1E-30]. Widening this range decreases the number of QR factorization steps required to cause the SVD to converge, in effect reducing the running time of the program. Wider zero ranges distort SVD, but might not negatively impact retrieval performance of the index if the TDM is sufficiently sparse. The default value of this parameter was used in the current work. The final parameter is the floating-point

error correction. This value – 1E-6 by default – corrects for rounding error in all floating-point calculations in the SVD program. As with the zero range, the default value was used in the current work.

4.1.3 Document Clustering and Boundary Extraction

The document clustering task was accomplished by applying algorithms implemented in Karypis's Cluster Toolkit (CLUTO) [31]. A data format adaptor was written for conversion between the output format of Rohde's SVD program and CLUTO's vector input format. The adaptor simply scales the right singular vectors by the singular values and prints the resulting vectors to a new file.

Karypis's CLUTO package permits user specification of dozens of parameters. This section first outlines the parameter classes and then specifies the values chosen for this pipeline. The three parameter classes support clustering algorithm selection, report generation, and visualization options.

CLUTO provides six clustering algorithms and a mechanism for composing sequences of clustering operations. The algorithms have a common parameter set: similarity functions, cluster criterion functions, row and column models, and pruning functions. The algorithms are a mixture of agglomerative, partitioning, and graph-based methods. Two algorithms may be composed with the second operating upon the solution of the first; this operation is usually applied by first partitioning and then agglomerating clusters. Each algorithm is specialized at run time by specification of a parameter set. Similarity functions include vector cosine, correlation coefficients, Euclidean distance, and Jaccard coefficients (for graph-based methods). Many criterion functions are

provided including linkage classifiers, UPGMA, and a collection of probabilistic classifiers. Additionally, a preprocessing step to the clustering algorithms may be instructed to perform two operations: transform the vectors into a variety of coordinate spaces; prune vectors out of the input set when they are duplicates, zero vectors, or otherwise non-informative.

During the literature review process, the Chameleon clustering method was identified as the best clustering method for the IR pipeline. CLUTO provides the capabilities needed to implement the Chameleon method and was therefore configured to emulate it. Chameleon composes partitioning and agglomerative algorithms – specifically graph-based methods. Early designs mimicked the graph-based method and determined that they were prohibitively slow on high-dimensional vectors. Instead, a composition of repeated bisection and hierarchical agglomeration was chosen. The partitioning step was specialized to use the correlation coefficient similarity function and the I2 criterion function. The agglomeration algorithm was specialized to use the correlation coefficient similarity function as well with the UPGMA criterion function. No coordinate transformation was applied to the vectors. The vector columns were pruned until 90% of the similarity between vectors remained. (According to Karypis, this pruning improves the processing time of the clustering algorithms without reducing the solution quality [31].) A more detailed journal of the process leading to these parameter choices is included in the Appendix.

CLUTO provides extensive reporting facilities from raw text files to formatted input suitable for a number of visualization systems. CLUTO's most essential output is a

report of the cluster sizes. Additional details include internal and external cluster similarities, z-scores representing the membership probabilities, feature analyses, and cluster labeling. Enhanced reports include detailed statistical summaries of each cluster's salient features, visualizations of clustering solutions, agglomeration trees, and more options that will be elided.

The basic reporting features were the most useful to the current work. CLUTO was instructed to output a file containing the cluster membership information for each vector. Z-scores for each vector are also emitted into the cluster solution file to support off-line analysis of the solution. Additional summary details were also requested as screen output to facilitate the parameter selection process detailed in the Appendix.

4.1.4 Surface Representation

Two surface representations were selected for implementation – axis-aligned bounding boxes and bounding spheres. These two were chosen from a wider field due to their simple data structures and efficient intersection tests. Both surface representations were derived from the cluster membership judgments output from CLUTO. Bounding boxes were constructed by recording the minimum and maximum values along each axis over all the members of a cluster. Bounding spheres were constructed by computing the centroid of each cluster's bounding box and constructing a radius length as half the length of the line segment between the minimum and maximum corners of the associated bounding box.

The separation of the bounding surfaces was computed to test the viability of applicability of the two representations to the IR problem. It was discovered that the

bounding spheres overlapped each other extensively to the point that a ray intersecting any one bounding sphere was likely to intersect them all. Investigation into this phenomenon revealed that the document clusters were roughly pencil-shaped – extended along rays radiating away from the origin. Bounding boxes, however, were found to provide perfect separation of the clusters. This result for axis-aligned boxes was unexpected. Natural language documents may be expected to cover a small set of topics; the vectors representing such a varied collection would tend to be bounded best by an oriented box. The choice of axis-aligned bounding boxes was motivated in part by the expectation that the extreme points of the boxes surrounding similar but distinct clusters would overlap and provide a mechanism for retrieving document records from a small neighborhood of similar clusters. The specificity of documents in the MEDLINE collection, however, causes the clusters to form around single index term axes instead of around arbitrary rays. This turned out to be both a blessing and an impediment as will be discussed in the next chapter.

A third cluster representation was designed around the signed distance field (SDF) concept in which each point in a data set is represented by its displacement relative to a convenient splitting plane. This improves on the bounding box and sphere representations which represent their members with full-rank vectors. Further the SDF stores most of the information needed to compute the similarity between a query vector and members of the cluster; the distance of a document to the splitting plane is linearly related to the similarity of the document vector and splitting plane (i.e. the ray containing the nearest point on the splitting plane). However, construction of a SDF requires the

ability to find an appropriate splitting plane and although the properties of such a plane are known for the IR problem – parallel with the centroid vector, perpendicular to the most separating axis – the fulfillment of these requirements is non-trivial in K dimensions. The idea of implementing cluster representations as a combination of bounding surfaces and signed distance fields is exciting, but was set aside as a subject for future work.

4.1.5 Query Service

Query service was designed from the start as a ray tracing engine adapted for operation in high dimensional spaces. A "query tracer," the name chosen for IR ray tracers, was implemented for this stage of the pipeline to perform intersection testing between rays derived from query vectors and the clusters constructed in earlier stages. The query tracer: (1) searches the document clusters to find those that intersect (or partially intersect) with the query ray; (2) ranks the data vectors in any intersected clusters in order of their similarity to the query ray; and (3) ranks the non-intersected clusters in order of their similarity to the query ray.

Once the design process of the cluster surface data structures was complete, the adaptation of the ray tracer was actually trivial. Most of the operations required were identical to traditional ray tracing requests; a query operation was added to make a distinction between raw and processed intersection test results. A framework of interface classes was designed to simplify implementation and extension of query tracers. Classes implementing the IR pipeline were then composed atop this interface and the Boost uBlas [46] linear algebra library. Because the Boost uBlas implementation is tuned for dense

matrix and vector operations, sparse versions of a few common operations such as vector-matrix multiplication and matrix multiplication were implemented as well.

Operation of the query tracer is described below. The query tracer loads each cluster of document vectors into a distinct and unique object in the search space. Each cluster object contains a collection of its document vectors. When a query message is received by a cluster, the query ray is tested against the boundaries of the enclosing surface. If the ray intersects the surface boundaries, then a list of document similarities is constructed and returned to the calling object. Cluster objects are contained by a scene graph that determines the search order and which of the clusters will be searched at all. The scene graph chosen for the current work imposes no order on the search and permits searching of all clusters. In this work, document clustering alone is sufficient to provide sub-linear search complexity in the number of documents. Other scene graph implementations made possible by the query tracing framework classes facilitate further reduction in search complexity, for example by implementing an R* tree over the document clusters. Scene graphs are contained by scene objects that process query results before passing them back to the user-level program. The IR scene graph imposes limits on the number of results returned in response to a query and ensures that results are returned in order of decreasing similarity. Variations on each of the objects described above are facilitated by the query tracing interface classes discussed above.

The query tracer implemented in SCRIBE does not directly serve natural language queries. The task of translating user queries into vectors in the semantic space is accomplished by a sequence of programs that stem the queries, construct index term

occurrence vectors, and produce pseudo-documents from the query term vectors. The decision to translate the queries offline was motivated by the need to reduce the memory load of the query tracer. Since the current query tracer implementation still loads the full-rank vector for each document into memory, the overhead of additionally loading the translation matrices restricts the ability of the query tracer to locate accurate matches in a reasonable amount of time. Both SCRIBE and the LSI system perform this offline query translation, so the process has no impact on the performance results.

4.2 Implementation of the Latent Semantic Indexing System

Latent semantic indexing was implemented for comparison to the SCRIBE system. This implementation was accomplished by configuring SCRIBE's query tracer to load each document vector into a distinct and unique object in the search space. The query operations were slightly modified to treat the entire scene graph as a single cluster in the query tracer. That is, SCRIBE only performs direct comparison of documents to the query vector when the query intersects the documents' enclosing surface, but LSI compares every document to the query vector. This behavior was implemented in the unordered scene graph class, thus causing a split of the LSI and query tracing source trees. Future work will obviate this change either by refactoring the class hierarchy to remove the distinction between clusters and document or by designing a scene graph tailored specifically to LSI operations. This divergence in behavior, though accomplished in a less than elegant fashion, is fundamental to the differences in the query tracing and LSI methods that are under scrutiny in this work.

CHAPTER V

EXPERIMENTAL RESULTS AND ANALYSIS

This chapter discusses experiments designed to test the following hypothesis:

Latent semantic indexing is an effective information retrieval technique, but search efficiency scales poorly and storage needs are unacceptable for large data collections. Techniques derived from machine learning and computer graphics can be applied to semantic indices to support more efficient search while maintaining comparable retrieval performance.

The following sections detail the experimental protocol used to gather performance data from both a Latent Semantic Indexing system and a SCRIBE system as described in Chapter V, the data resulting from the experiment, and statistical analyses of the performance data.

5.1 Data Sets

W.R. Hersh's OHSUMED [30] document collection, queries, and relevance judgments distributed by NIST in association with the TREC-9 filtering track were chosen as the input for these experiments due to their availability and the author's familiarity with their data formats. OHSUMED includes sixty-three queries and associated relevance judgments for training and test collections of MEDLINE abstracts spanning the years

1987 and 1988-91 respectively. The training collection contains 54,710 documents. The test collection contains 293,856 documents.

5.2 Metrics for Measuring Success

The following retrieval and performance metrics were collected:

- Recall
- Precision
- F-measure
- Running time

The measurements of the prototype pipeline were compared to measurements of the author's LSI implementation on the same data set. Recall, precision, and f-measure for the prototype were expected to be similar to the measurements for LSI. Running time was expected to be lower for SCRIBE than LSI.

5.3 Experimental Design

5.3.1 Hypothesis

An experiment was performed to compare the performance of SCRIBE with LSI. The CPU time, recall, precision, and f-measure metrics were chosen as indicators likely to provide insight into the relationships between the systems' performance characteristics. The hypothesis stated above includes an expectation that SCRIBE will improve upon the efficiency of LSI while maintaining comparable retrieval capability. With respect to the indicator metrics this implies that:

(i) Total CPU time of a SCRIBE query is less than the time for an LSI query with the same parameters;

(ii) Recall, precision, and f-measures of a SCRIBE query are no worse than for an LSI query with the same parameters.

5.3.2 Experimental Protocol

A protocol was designed to collect measurements of the performance metrics of a SCRIBE system and compare these data to those collected from an identically configured LSI system. Each experiment exercised both a SCRIBE and an LSI system over identical document collections, queries and relevance judgments, indexing terms, and dimensionalities. The experiment was repeated with four document collections created by randomly sampling 25% of the OHSUMED test collection. In each experiment, measurements of the performance metrics were taken for each of the sixty-three queries provided with the OHSUMED collection. The resulting four measurements for each query were averaged across the four experiments (to reduce noise introduced by uneven sampling of judged documents across the queries) yielding sixty-three data points per metric.

5.4 Results

Measurements of the SCRIBE and LSI processes are listed in the Appendix. Two listings are provided for each process corresponding to the average measurements of the performance metrics over the four document collections sampled from the OHSUMED collection. The results are organized into spreadsheets with columns reporting the timing

and retrieval statistics previously discussed. OHSUMED relevance judgments are documents chosen by subject matter experts who searched the MEDLINE archive for documents fulfilling the information need stated in each query. Because the OHSUMED collection was randomly sampled to create the four test collections, the number of available relevance judgments varies between experiments; however, each sampled document collection contains ten relevance judgments per query on average. Because the number of documents retrieved is independent of the number of relevance judgments, running time is unaffected by the variance in the number of judged documents. The performance metrics chosen for these experiments are sensitive to variations in the number of judged documents; all three performance metrics are ratios of the number of judged documents and some other feature. To account for variations in the performance metrics due to sampling, measurements at each data point were averaged across the four document collections. The averaged measurements are summarized in Figures 5.1 – 5.4.

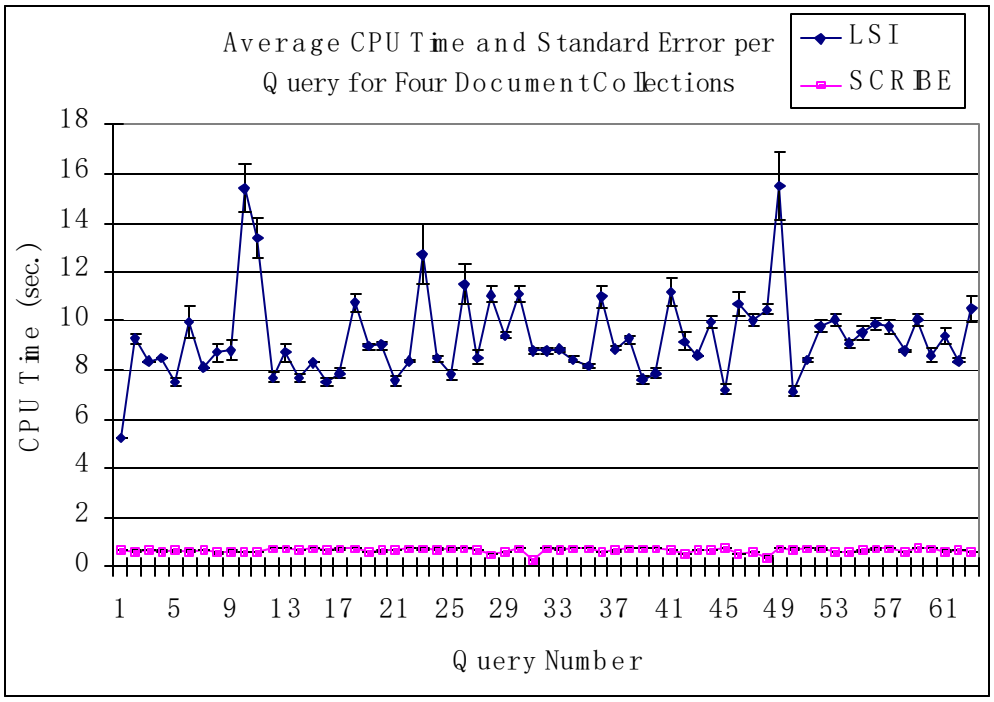

Figure 5.1 Running time summary

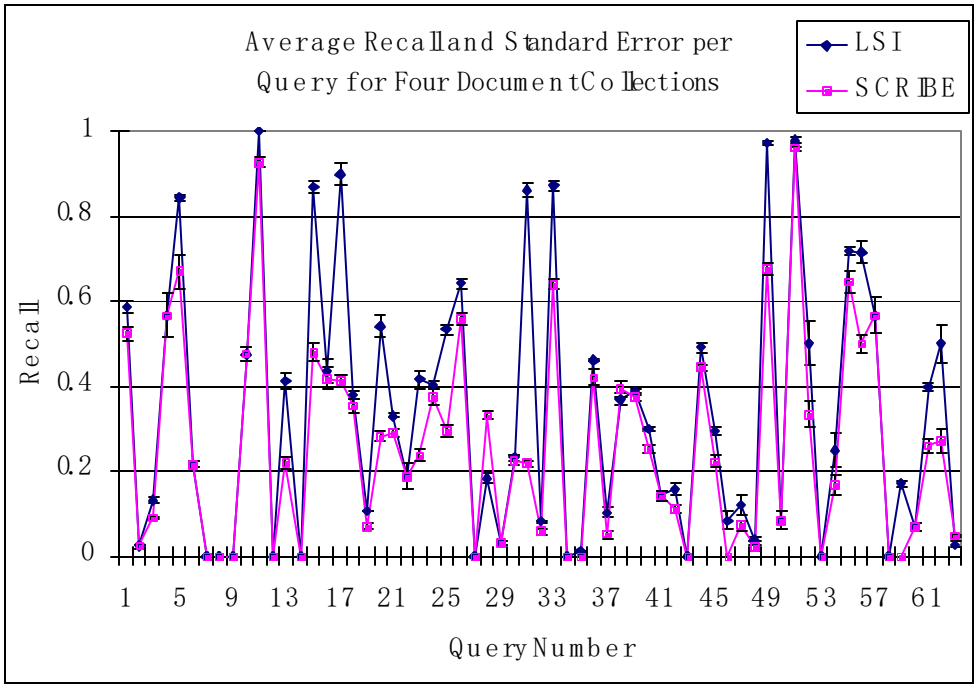

Figure 5.2 Recall summary

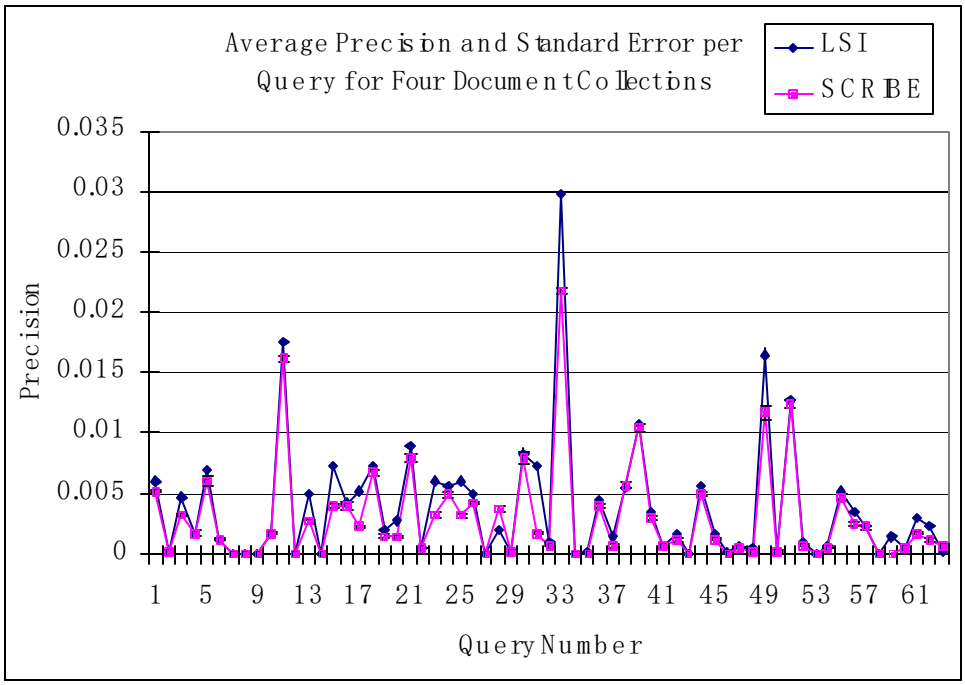

Figure 5.3 Precision summary

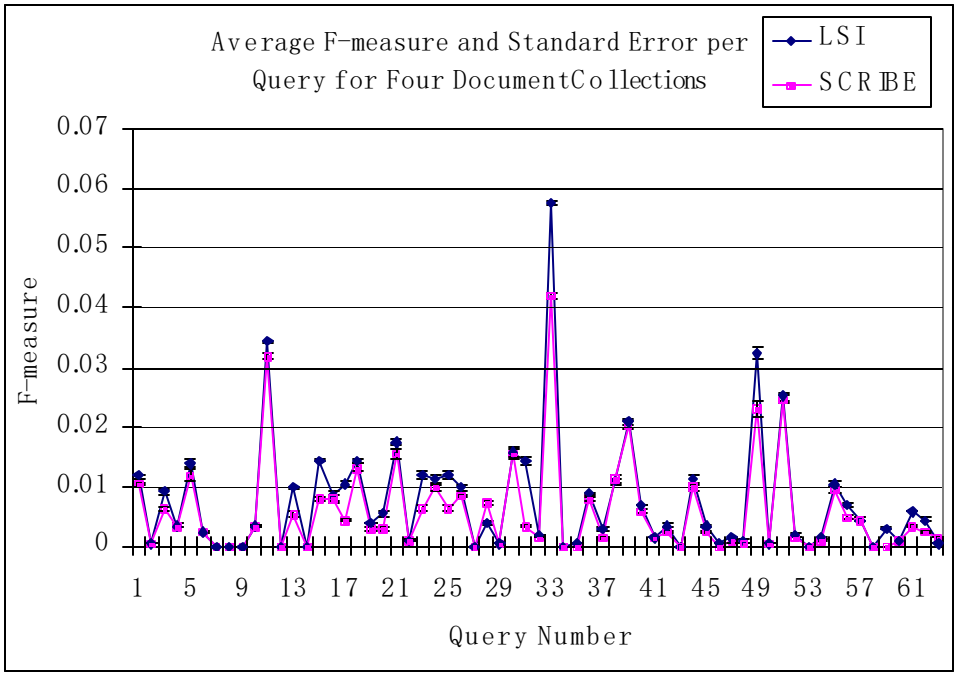

Figure 5.4 F-measure summary

5.5 Analysis

The results were analyzed using hypothesis testing by statistical methods. Initially, the population means of each metric were examined with a Mann-Whitney rank sum test. After failure to reject the null hypothesis for three of the metrics, the characteristics of data supporting the null hypothesis were considered. A power analysis and an ANOVA of means were performed to examine the effects of algorithm choice on recall, precision, and f-measure. This section summarizes the details of each test and presents an interpretation of the results.

A Mann-Whitney rank sum test [47] was performed for each metric to test the null hypothesis that the measurements from both the LSI and SCRIBE systems were drawn from the same population. Probability tables for the Mann-Whitney U statistic are not calculated for sample sizes greater than twenty-five. Instead, z-scores are calculated from the U statistic and approximations from the normal distribution are used instead. Herrnstein's ρ statistic [47] was calculated as well. ρ is an effect size measure representing the amount of overlap between two sampling distributions. ρ ranges from 0.0 to 1.0 with 0.5 indicating complete overlap and the extremes indicating no overlap. ρ is derived from the Mann-Whitney statistic as $\frac{U}{n_1 * n_2}$ where U is the Mann-Whitney statistic and n_1 and n_2 are the number of samples in the two groups.

CPU time populations for the two algorithms were found to be totally disjoint with a ρ statistic greater than 0.99 and a z-score $(z = 9.20)$ indicating a probability less than 0.001 that the two sets of measurements were drawn from the same population. This

finding is consistent with Figure 6.1. Therefore, the null hypothesis is rejected for CPU time in favor of the alternative hypothesis: SCRIBE and LSI exhibit significantly different running times during query service. Moreover, SCRIBE exhibits significantly lower running times than LSI. On average SCRIBE returns query results 8.6 seconds faster than LSI. This is a very large effect size – approximately thirty-four standard deviations $(d \cong 34)$.

No evidence was found to indicate rejection of H_0 with respect to the recall, precision, and f-measure statistics. Z-scores for these tests $(z = 1.29..1.57)$ indicated probabilities between 0.07-0.15 that the measurements were drawn from the same population. The ρ statistics for these three metrics ranged from 0.570-0.585 indicating that the LSI and SCRIBE populations are similar for each metric. It was decided that an analysis of variance (ANOVA) should be performed to further explore the data.

An *a priori* power analysis of the one-way ANOVA of means was performed to predict the sample size needed to achieve a statistical power level of 0.8 for several effect sizes. The statistical power is derived as $1 - \beta$ where β is the probability of committing a Type II error – failing to reject a false H_0. M. Friendly's `fpower` SAS macro [25] was used to calculate *a priori* power levels for several effect sizes (in terms of standard deviations) over a range of sample sizes including the number of data points in the current experiment. It was determined that with sixty-three samples effects of at least 0.5 standard deviations can be detected with 80% power and effects of at least 0.8 standard deviations with 99% power. That is, the sample size of the current experiment (N=63) is sufficiently large to provide an 80% likelihood of rejecting any false null hypothesis with

an effect size of 0.5 standard deviations or larger $(d \geq 0.5)$. As the effect size increases, the probability of accepting a false H_0 decreases. Placing strict bounds on statistical power allows us to accept H_0 when $P(X \mid H_0) > 0.5$ at the α=0.5 level with a reasonable degree of assurance that H_0 is actually true. It was decided that an effect size of 0.5 standard deviations represented the smallest effect likely to be significant. Examination of the data from this experiment reveals that a change of 0.5 standard deviations is equivalent to adding one relevant document in ten, on average, to the raw recall measurements from either algorithm or one document in a thousand to the precision scores of either algorithm. We assume that such a small difference in performance is barely significant. With this in mind, we fix power at 80% and proceed to analyze the ANOVA.

A one-way ANOVA relating the average measurements of each metric to algorithm choice was performed over all the data points (N=63). As discussed above, the experiment was powerful enough to detect a false H_0 80% of the time (N=63, d=0.5, power=80%). Between groups F tests relating performance metrics to algorithm choice are summarized in Table 6.1. No significant difference was observed between algorithms for recall, precision, or f-measure at the α=0.05 level, and we can bound any possible effect at $d \leq 0.5$ standard deviations (e.g., less than ± 1 document out of 1000 for recall) for α=0.05, N=63, and power=0.80.

Table 5.1 Summary of ANOVA of means relating metrics to algorithms.

ANOVA of Metrics Between Algorithm Groups
(N=63, α=0.05, power=0.8)

	F(1,124)	$P(X\|H_0)$	*d* bounded	*d* observed
Recall	2.543	.113	0.50	0.28
Precision	.931	.336	0.50	0.17
F_measure	.949	.332	0.50	0.17

In summary, SCRIBE and LSI were found to exhibit significantly different running times under the Mann-Whitney rank sum test. Specifically, SCRIBE's average running time was found to be an order of magnitude lower than average times for LSI. An ANOVA power analysis bounded any possible effect size for recall, precision, and f measure to $d \leq 0.5$ standard deviations with power=80%, N=63, α=0.05, and we determine that $d \leq 0.5$ standard deviations is too small of an effect to meaningfully favor one algorithm over another. On the strength of that evidence, we accepted that SCRIBE and LSI exhibit the same retrieval performance.

CHAPTER VI

CONCLUSIONS AND FUTURE WORK

This chapter summarizes the findings that SCRIBE improves upon LSI's running time by a factor of ten without impacting retrieval performance. Directions of future research are discussed. Two SCRIBE enhancements are outlined: parameterization of clusters' overlapping regions and reduction of the memory required to store a SCRIBE index. Human computer interface topics discussed include semantic cluster labeling and visualization of query results.

6.1 Summary of Results

SCRIBE is an information retrieval system based on an algebraic vector model of information retrieval similar to Latent Semantic Indexing. SCRIBE enhances the algebraic vector model of IR with clustering techniques borrowed from machine learning. SCRIBE packages its semantic index into efficiently searchable computer graphics data structures adapted for high-dimensional duty. While designing SCRIBE, a generic information retrieval framework emerged. This framework abstracts the fundamental stages of information retrieval so that a complete SCRIBE system can be constructed from pluggable modules. In the course of this research, both LSI and SCRIBE have been implemented as modules in this IR framework.

Experiments performed during this research show that SCRIBE and LSI perform equally well with respect to three important IR metrics. Statistical hypothesis testing over the ANOVA F statistic indicates that SCRIBE and LSI perform equally well with respect to the recall, precision, and f-measure metrics. Both algorithms exhibit similar levels of recall; they find the same number of relevant documents in response to a user's request. Likewise, SCRIBE's and LSI's responses are similarly precise; the algorithms are equally robust against noise. The f-measure statistic – derived from recall and precision – also demonstrates identical behavior for SCRIBE and LSI. The only significantly different performance measure between the two algorithms is running time. Hypothesis testing over the Mann-Whitney U statistic indicates that SCRIBE responds to queries more than ten times faster than LSI. This improvement is bought at the cost of a longer index creation process, but the overhead of clustering and surface building is negligible compared to the runtime savings during query service.

6.2 Contributions

In this research we developed SCRIBE, an IR application demonstrating a clustering method for organizing and searching large document collections. SCRIBE includes a generic framework for developing new information retrieval systems. SCRIBE's search algorithm, based on ray tracing, provides time-efficient fulfillment of users' information needs. Finally, SCRIBE's data structures will facilitate visualization and exploration of large document collections.

6.3 Future Research

Three proposed aspects of the SCRIBE system were not implemented in the current work: storage charge reduction, visualization, and labeling. Work on these components is ongoing and is discussed in the following sections. A need to parameterize cluster overlap regions was discovered during implementation and testing of SCRIBE and is also a topic of continuing research.

6.3.1 Cluster Overlap Parameterization

Axis-aligned bounding boxes (AABBs) were chosen as the surface representation of SCRIBE clusters in part because they do not tightly approximate the shape of arbitrarily aligned document clusters. Figure 7.1 shows a group of document clusters indexed by the terms computer and art. The portion of each cluster near the origin of the coordinate system represents documents in which the index terms appear with relatively low frequency. Foltz [23] describes documents in this region as relatively general in scope – containing many index terms, but not using any of them more frequently than the others. Note that the low-frequency extremities of the AABBs of the clusters overlap such that a query vector intersecting at least one of the clusters has a high probability of intersecting several of them. This configuration is typical of the clusters that were thought to exist in the OHSUMED collection prior to the experiments previously described. Figure 7.2 shows the configuration that is now thought to be more typical of the clusters in the OHSUMED collection. Because the clusters tend not to overlap, SCRIBE sometimes ignores relevant documents because the enclosing cluster just barely fails to intersect with the query vector.

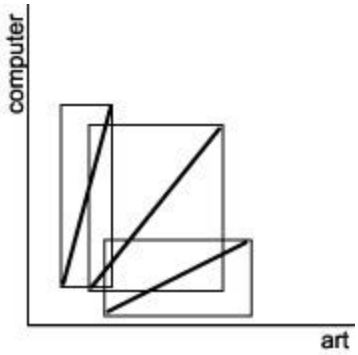

Figure 6.1 Expected cluster topology

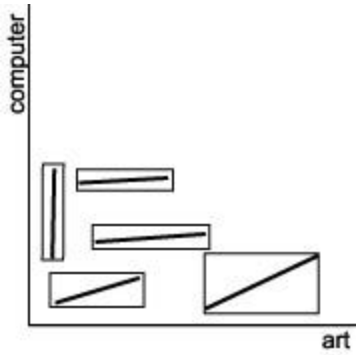

Figure 6.2 Actual cluster topology

We have considered several potential solutions to the near miss problem involving parameterization of the cluster intersection test. Clusters may be instructed to detect intersection with queries passing through a region near the boundary surface, effectively expanding or shrinking their volume in all directions. Conversely, the return value of the intersection test – a Boolean in the current implementation – may take on a value from a range of levels to indicate degrees of similarity with a cluster. In this case, a cutoff parameter supplied at run time would determine the similarity at which clusters intersect with query vectors. Additional research of this topic will include a study of the makeup of document collections in which this sort of parameterization is really needed. For example, we do not expect that the texts of a public library would contain as few low-frequency entries as the OHSUMED collection. The ability to adjust for cluster density variations will become important as novel data types become the focus of IR research.

6.3.2 Storage Charge Reduction Using Signed Distance Fields

The current implementations of SCRIBE and LSI operate on exactly the same data and load the same amount of data into memory at run time. Our proposal to reduce

the memory requirements was founded on an expectation of adapting Signed Distance Fields (SDF) to high-dimensional spaces. We have since learned that those assumptions were quite naïve.

To construct a SDF from a SCRIBE cluster, the normal vector of a splitting plane is calculated at the center of the cluster and perpendicular to the axis along which the document members vary most. In three dimensions the splitting plane normal is found using the cross-product operator on two convenient vectors. Unfortunately, the cross-product is not defined for vectors of arbitrarily high dimension. However, an equivalent of the cross-product can be found in k dimensions when k-1 basis vectors are known. The subject of this research involves formalizing a method for choosing basis vectors that approximate the splitting plate of a k-dimensional cluster.

6.3.3 Cluster Labeling

Cluster labeling is a key requirement for integration of SCRIBE into a machine learning system, and also for construction of user interfaces atop SCRIBE. Without labels, the contents of SCRIBE clusters cannot be automatically recognized by a machine learning algorithm. Likewise, users must be presented with a summary of a cluster's contents if SCRIBE is to be made into a useful IR system. Unsupervised cluster labeling algorithms suitable to this task have begun appearing in the literature over the past few years. Future research will explore the feasibility of adapting these algorithms into SCRIBE modules. Cluster labeling will enable the incorporation of relevance feedback mechanisms into SCRIBE.

6.3.4 Visualization

Visualization of the SCRIBE clusters was not a primary goal of this research. Since SCRIBE's data structures and search algorithms were adapted from ray tracing, though, partial visualization support has already been integrated into the SCRIBE query engine. Future work in visualization of query results and the SCRIBE index depends upon the availability of some cluster labeling support. As a side effect of SCRIBE's principle component analysis the dimensions of document vectors are ordered so that picking the first k elements of all document vectors produces the maximally visible, k-dimensional projection of the vector space. Visualization in two or three dimensions is essentially built in to SCRIBE by default. Two additional items must be added to fully enable visualization of SCRIBE's clusters. First, the cluster intersection tests must be specialized to operate on query vectors with any dimensionality less than or equal to the dimensionality of the index. Second, a representation of material properties must be added to SCRIBE's document and cluster objects. A rudimentary material properties class was built into the SCRIBE's Object base class. Operations on the material properties have not yet been exercised on the objects returned in SCRIBE's query responses.

Future research will involve testing the existing visualization mechanisms in SCRIBE and adding new capabilities as necessary. Construction of an IR tool is a logical next stage of development. User studies comparing SCRIBE-based visualizations to existing IR tools will be performed in this phase of development.

REFERENCES

[1] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide*, 3 ed. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1999.

[2] R. K. Ando, "Latent Semantic Space: Iterative Scaling Improves Precision of Inter-document Similarity Measurement," presented at 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Athens, Greece, 2000.

[3] R. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*, 1 ed: Addison-Wesley, 1999.

[4] M. W. Berry, "Large Scale Singular Value Computations," *International Journal of Supercomputer Applications*, vol. 6, pp. 13-49, 1992.

[5] P. E. Black, "BSP-tree," in *Dictionary of Algorithms and Data Structures*, 2005.

[6] P. E. Black, "k-d tree," in *Dictionary of Algorithms and Data Structures*, 2005.

[7] A. Borodin, R. Ostrovsky, and Y. Rabani, "Subquadratic Approximation Algorithms for Clustering Problems in High Dimensional Spaces," presented at Thirty-first Annual ACM Symposium on Theory of Computing, Atlanta, GA, 1999.

[8] A. Broder, S. Glassman, M. Manasse, and G. Zweig, "Syntactic Clustering of the Web," presented at Sixth International World Wide Web Conference, Santa Clara, California, 1997.

[9] C. Chen, N. Stoffel, M. Post, C. Basu, D. Bassu, and C. Behrens, "Telcordia LSI Engine: Implementation and Scalability Issues," presented at 11th International Workshop on Research Issues in Data Engineering: Document Management for Data Intensive Business and Scientific Applications, Hiedelberg, Germany, 2001.

[10] M. W. Davis and P. W. Foltz, "Learning Via Compact Data Representation," 1998.

[11] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman, "Indexing by Latent Semantic Analysis," *Journal of the Society for Information Science*, vol. 41, pp. 391-407, 1990.

[12] S. T. Dumais, "Enhancing Performance in Latent Semantic Indexing Retrieval," *Behavior Research Methods, Instruments and Computers*, vol. 23, pp. 229-236, 1991.

[13] S. T. Dumais, "LSI meets TREC: A Status Report," presented at The First Text REtrieval Conference (TREC1), Gaithersburg, MD, 1993.

[14] S. T. Dumais, "Latent Semantic Indexing (LSI) and TREC-2," presented at The Second Text REtrieval Conference (TREC2), Gaithersburg, MD, 1994.

[15] S. T. Dumais, "Using Linear Algebra for Information Retrieval," *SIAM Review: Society for Industrial and Applied Mathematics*, vol. 37, pp. 573-595, 1995.

[16] S. T. Dumais, "Using LSI for Information Filtering: TREC-3 Experiments," presented at The Third Text REtrieval Conference (TREC3), Gaithersburg, MD, 1995.

[17] S. T. Dumais, G. W. Furnas, T. K. Landauer, and S. Deerwester, "Using Latent Semantic Indexing to Improve Information Retrieval," presented at Proceedings of CHI'88: Conference on Human Factors in Computing, New York, 1988.

[18] G. Farin, "A History of Curves and Surfaces in CAGD," in *Handbook of Computer Aided Geometric Design*, G. Farin, J. Hoschek, and M.-S. Kim, Eds., 5th ed. North Holland: Elsvier, 2002.

[19] P. W. Foltz, "Comprehension, Coherence and Strategies in Hypertext and Linear Text," in *Hypertext and Cognition*, J.-F. Rouet, J. J. Levonen, A. P. Dillon, and R. J. Spiro, Eds. Hillsdale, NJ: Lawrence Erlbaum Associates, 1996.

[20] P. W. Foltz, "Latent Semantic Analysis for Text-Based Research," *Behavior Research Methods, Instruments and Computers*, vol. 28, pp. 197-202, 1996.

[21] P. W. Foltz, M. A. Britt, and C. A. Perfetti, "Reasoning from Multiple Texts: An Automatic Analysis of Readers' Situation Models," presented at 18th Annual Cognitive Science Conference, Lawrence Erlbaum, NJ, 1996.

[22] P. W. Foltz and S. T. Dumais, "Personalized Information Delivery: An Analysis of Information Filtering Methods," *Communications of the ACM*, vol. 35, pp. 51-60, 1992.

[23] P. W. Foltz, W. Kintsch, and T. K. Landauer, "The Measurement of Textual Coherence with Latent Semantic Analysis," *Discourse Processes*, vol. 25, pp. 285-307, 1998.

[24] A. R. Forrest, "Interactive Interpolation and Approximation by Bezier Polynomials," *The Computer Journal*, vol. 15, pp. 71-79, 1972.

[25] M. Friendly, "fpower: Power computations for ANOVA designs," 1.2 ed, 1995.

[26] I. Gath and A. B. Gev, "Unsupervised Optimal Fuzzy Clustering," *IEEE Transactions on Pattern Analysis and Machine Learning*, vol. 11, pp. 773-780, 1989.

[27] S. Guha, R. Rastogi, and K. Shim, "CURE: An Efficient Clustering Algorithm for Large Databases," presented at 1998 SIGMOD International Conference on Management of Data, Seattle, WA, 1998.

[28] D. Hand, H. Mannila, and P. Smyth, *Principles of Data Mining*, 1 ed. Cambridge, MA: MIT Press, 2001.

[29] D. Hearn and M. P. Baker, *Computer Graphics with OpenGL*, 3rd ed. Upper Saddle River, NJ: Pearson Prentice Hall, 2004.

[30] W. R. Hersh, C. Buckley, T. J. Leone, and D. H. Hickam, "OHSUMED: An interactive retrieval evaluation and new large test collection for research," presented at 17th Annual ACM SIGIR Conference, 1994.

[31] G. Karypis, "CLUTO: A Clustering Toolkit," 2.1.1 ed. Minneapolis, MN: University of Minnesota, 2003, pp. A collection of tools for clustering low- and high-dimensional data.

[32] G. Karypis, E.-H. S. Han, and V. Kumar, "Chameleon: Hierarchical Clustering Using Dynamic Modeling," *Computer*, vol. 32, pp. 68-75, 1999.

[33] T. Kohonen, *Self-Organizing Maps*, vol. 30, 3 ed. New York, NY: Springer, 2001.

[34] T. G. Kolda and D. P. O'Leary, "Computation and Uses of the Semidiscrete Matrix Decomposition," University of Maryland, April, 1999.

[35] D. Koller and M. Sahami, "Hierarchically Classifying Documents Using Very Few Words," presented at 14th International Conference on Machine Learning, 1997.

[36] T. K. Landauer, P. W. Foltz, and D. Laham, "An Introduction to Latent Semantic Analysis," *Discourse Processes*, vol. 25, pp. 259-284, 1998.

[37] T. K. Landauer, D. Laham, and P. W. Foltz, "Learning Human-like Knowledge by Singular Value Decomposition: A Progress Report," *Advances in Neural Information Processing Systems*, vol. 10, pp. 45-51, 1998.

[38] A. McCallum, K. Nigam, and L. H. Ungar, "Efficient Clustering of High-dimensional Data Sets with Application to Reference Matching," presented at Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Boston, MA, 2000.

[39] M. Porter, "Porter Stemming Algorithm," 1 ed: tartus.org, 2006.

[40] D. Rohde, "SVDLIBC," 1.34 ed. Boston, MA: TedLab, Massachusettes Institue of Technology, 2005.

[41] G. Salton and C. Buckley, "SMART version 11 English stopword list," 11 ed: Cornell University, 1999.

[42] P. Shirley, *Fundamentals of Computer Graphics*, 1 ed. Natick, MA: A. K. Peters, 2002.

[43] N. Slonim and N. Tishby, "Document clustering using word clusters via the information bottleneck method," presented at 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Athens, Greece, 2000.

[44] C. J. van Rijsbergen, *Information Retrieval*, 2nd ed. Glasgow, Scotland: University of Glasgow, 1979.

[45] K. Versprille, "Computer Aided Design Applications of the Rational B-spline Approximation Form." Syracuse, NY: Syracuse University, 1975.

[46] J. Walter and M. Koch, "uBLAS: Boost C++ Libraries Basic Linear Algebra System," Boost 1.33.0 ed: Boost.org, 2002.

[47] Wikipedia, "Mann-Whitney U," vol. 2006: Wikipedia.org, 2006.

[48] P. Willett, "Recent Trends in Hierarchic Document Clustering: a Critical Review," *Information Processing and Management: an International Journal*, vol. 24, pp. 577-597, 1988.

[49] M. B. Wolfe, M. E. Schreiner, B. Rehder, D. Laham, P. W. Foltz, W. Kintsch, and T. K. Landauer, "Learning from Text: Matching Readers with Texts by Latent Semantic Analysis," *Discourse Processes*, vol. 25, pp. 309-336, 1998.

APPENDIX A

CLUSTER PARAMETER SEARCH JOURNAL

APPENDIX A

CLUSTER PARAMETER SEARCH JOURNAL

Notes on the Development of Document Vector Clustering Parameters

I searched for a clustering solution that met these needs:

1) Maximize the internal similarity of each cluster's members;
2) Minimize the number of clusters;
3) Minimize the running time of the clustering algorithm.

These requirements were prioritized in the order they are listed above. My search strategy relied upon three metrics - each related to the corresponding requirement listed above:

1) Average similarity - the ratio of the total internal similarity of all clusters divided by the number of clusters in the solution.
2) Number of clusters.
3) Running time of algorithm.

I must note that my evaluations were informal and I did not retain my measurements once I was satisfied with my parameter set. I could probably regenerate the data upon which I based my decision if necessary.

I began by choosing a few algorithms that were likely to yield meaningful document clustering solutions as indicated by previous work in the field of research (mostly gleaned from the CLUTO documentation and companion publications). I chose to examine the speed of these algorithms first:

- Graph partitioning,

- Graph partitioning followed by agglomerative,
- Direct k-means (incremental learning of N cluster parameters),
- Bisecting k-means,
- Bisecting k-means followed by agglomerative.

I instructed the clustering program to produce 1000-way clustering solutions for each of the above methods over the ohsu.87 MEDLINE data set. Graph partitioning and direct k-means methods ran for prohibitively long times and were eliminated from consideration. Bisecting k-means with and without agglomeration both ran in an acceptable time frame.

I next compared the average similarity of 400-way solutions from bisecting k-means with and without agglomeration (for agglomeration the bisection was carried to 1000 partitions which were then re-merged). The agglomerated solution tended to improve the average similarity over a variety of tangential parameter sets including agglomerative criterion function and bisection criterion function. Thus, I chose to optimize the number of clusters only for bisecting k-means with agglomeration.

I generated bisecting k-means with agglomerations solutions described by the ordered pair (300, 1000), (400, 1000), (600, 1000), (1000, 2000), and (1000, 5000) where the first number is the size of the agglomeration solution and the second number is the size of the bisecting k-means solution. For each of these solutions the agglomerative criterion function was UPGMA and the bisection criterion function was I2 as described in Karypis, 2003. The pair (400, 1000) was found to maximize the average similarity

among the selected tuples. No further minimization of the number of clusters was attempted.

At this point I had obtained a parameter set {bisecting k-means with agglomeration, (400, 1000)} which described a clustering strategy that optimized the metrics previously chosen as guides to a preferred solution. Intermediate results were deleted to avoid accidental corruption of experimental data. All four data sets were clustered according to the new parameter set.

A Note About the Clustering Solutions

Each of the four clustering solutions contains a "junk" cluster that appears to contain most or all of the outliers from the document collection. These outlying points are given a very low internal similarity score in terms of the correlation coefficient metric that was used during clustering. However, they have a markedly consistent structure. Form a dense cloud within or very near the unit sphere surrounding the origin of the semantic coordinate system. This positioning indicates the presence of a group of documents that contained so few distinctive indexing terms that they were marginalized during the singular value decomposition. One semantic interpretation of this position is that the documents contained little or no information. Another interpretation is that each of the documents is so unique that no other documents in the collection share with it any semantic relationship. It is expected that the documents in these "junk" clusters represent a mixture of the two situations. In either case, our ability to address the semantic content of the members of these clusters was lost during the knowledge inference step. A

possible solution to this loss of information is the inclusion of metadata such as the MESH indexing terms in the indexed text of each document.

APPENDIX B

EXPERIMENTAL PROCEDURE

APPENDIX B

EXPERIMENTAL PROCEDURE

The following steps should be performed in a Cygwin environment.
The exp_tools/ directory should be in the PATH.

1. Use "ohsustrip" from "ohsu_strip.cpp" to produce reduced ohsumed.xx files.
   Reduced files only include the pertinent fields - .I, .U, .W - from the original document collection.
   Reduced files are about 30% smaller than the originals.
   $ ohsustrip ohsumed.xx ohsumed_r.xx

Optional: Sample a percentage of the document collection instead of the whole collection.
   This is useful when the collection is too large to be addressed as a contiguous term document matrix in main memory. The svd program will emit an error when the collection is too large.
   $ ohsustrip ohsumed.xx ohsumed_rNN.xx 0.NN
      NN is the sample rate.

2. Use "stem" from C++ port of Porter's algorithm to produce stemmed document collection.
   Stemmed files are uniformly lowercase and include stemmed abstracts only.
   Stemmed files are marginally <5% smaller than the reduced files.
   $ stem ohsumed_r.xx >ohsumed_rs.xx

3. Delete the intermediate oshumed_r.xx file. It is no longer needed and is very large.
   $ rm ohsumed_r.xx

3. Use "stem" to produce a stemmed stop list from SMART's "englishST.txt" list.
   $ stem englishST.txt >englishST_s.txt

4. Use "vectorizer" with the stemmed stop list to produce a term document matrix from the reduced and stemmed ohsumed.xx collection. (Running vectorizer with no arguments will produce a usage guide.)
   $ vectorizer ohsumed_rs.xx englishST_s.txt

5. Use "svd" to convert tdm.sparse from a sparse text to a sparse binary file.
   Sparse binary files are about 50% smaller than sparse text files.
   $ svd -r st -w sb -c tdm.sparse tdm.sparse.bin

6. Delete the tdm.sparse text file. The sparse binary file is a one-to-one equivalent but takes half the space to store and runs through the SVD process more quickly.

7. Rename tdm.sparse.bin resulting from svd to something informative.
   $ mv tdm.sparse.bin ohsu_88-91_rst.tdm.sparse.bin

8. Save TDM file, row_template.txt, and doc_id_records.txt together in a separate directory.
   $ mv doc_id_records.txt tdm_files/
   $ mv row_template.txt tdm_files/
   $ mv ohsu_88-91_rst.tdm.sparse.bin tdm_files/

Optional: Compress the tdm file with gzip
   $ gzip --best ohsu_88-91_rst.tdm.sparse.bin

9. Use "svd" to produce the decomposition of the ohsu term document matrix. svd can produce the subset of the decomposition corresponding to just the N greatest eigenvalue triples by specifying N to the -d option. Get the 300-dimensional approximation of the TDM in dense binary format matrix files
   $ svd -d 300 -o svd_files/ -r sb -w db -v 2 tdm_files/ohsu_88-91_rst.tdm.sparse.bin.gz

10. Compress the matrix files in svd_files/ for storage.
    $ tar czvf ohsu_88-91_rst.svd.tgz svd_files/*
    $ mv ohsu_88-91_rst.svd.tgz svd_files/

11. Use "right_singular_scale" to find the product of S * Vt and transpose the result into the format expected by CLUTO.
    $ cd svd_files
    $ right_singular_scale -S -Vt V_scaled.txt

Optional: compress the V_scaled.txt file. This is recommended. Decompress the file for use with CLUTO, but normally keep it compressed.
   $ gzip --best V_scaled.txt

12. Copy the uncompressed document vector matrix and document ID files to the cluster_files directory
    $ cp svd_files/V_scaled.txt cluster_files/V_scaled.mat
    $ cp tdm_files/doc_id_records.txt cluster_files/doc_id_records.rlabel

13. Use "vcluster" from the CLUTO package to perform repeated bisecting k-means clustering to 1000 clusters followed by agglomerative clustering of the k-means solution to 600 clusters. Use correlation coefficients to measure document vector similarity. Choose the 'best' dimension along which to partition k-means clusters. Use the UPGMA criterion function for agglomerative clustering. Do not scale the rows or columns. Prune the columns to account for 90% of the similarity between documents (improves running time without negatively impacting clustering solution). Save output to vcluster.txt.

```
$ cd cluster_files/
$ ~/exp_tools/cluto-2.1.1/Win32/vcluster -clmethod=rb -sim=corr \
    -agglocrfun=upgma -agglofrom=1000 -cstype=best \
    -rowmodel=none -colmodel=none -colprune=0.9 \
    -clustfile=V_scaled.mat.clustering.rb.1000.agglo.600 \
    -rlabelfile=doc_id_records.rlabel -zscores V_scaled.mat 600 >vcluster.txt
```

14. Use "organize_clusters" to produce the extracted clusters from the document collection. Save the output to organized_clusters.txt.
    ```
    $ ~/exp_tools/organize_clusters.exe V_scaled.mat doc_id_records.rlabel \
        V_scaled.mat.clustering.rb.1000.agglo.600 600 organized_clusters.txt
    ```

15. Use "build_surfaces" to produce bounding boxes and bounding spheres that enclose the members of each of the organized clusters. Save the output to bounding_boxes.txt and bounding_spheres.txt respectively.
    ```
    $ ~/exp_tools/build_surfaces.exe organized_clusters.txt 600 300 bounding_boxes.txt
        bounding_spheres.txt
    ```

16. Use "stem" and a text editor to stem the query input file, "query.ohsu.1-63". Save the output to query_stemmed.ohsu.1-63.
    ```
    $ ~/exp_tools/stem.exe query.ohsu.1-63 >> query_stemmed.ohsu.1-63
    ```
    (Use text editor to repair XML tags, "Number:" tags, OHSU query identifiers, and "Description:" tags.)

17. Use "scale_queries" to produce query pseudo-documents from the stemmed query records. Save the output to query_vectors.txt.
    ```
    $ ~/exp_tools/scale_queries.exe query_stemmed.ohsu.1-63 svd_files/-Ut svd_files/-S
        tdm_files/row_template.txt query_vectors.txt
    ```

18. Use "querytracer" to produce a list of document IDs for documents that match the queries in query_vectors.txt. Save the output to query_matches_timing.txt
    ```
    $ querytracer.exe cluster_files/organized_clusters.txt \
        cluster_files/bounding_boxes.txt query_vectors.txt \
        600 300 >> results/query_matches_timing.txt
    ```

19. Use "lsitracer" to produce a list of document IDs for documents that match the queries in query_vectors.txt. Save the output to lsi_query_matches_timing.txt
    ```
    $ lsitracer.exe cluster_files/organized_clusters.txt cluster_files/bounding_boxes.txt \
        query_vectors.txt 600 300 >> results/lsi_query_matches_timing.txt
    ```

20. Use "align_qrels" to extract the query relevance records for only the documents in each experimental collection from among the records for the entire TREC OHSU corpus.
    ```
    $ align_qrels.exe qrels.ohsu.88-91 tdm_files/doc_id_records.txt \
    ```

```
    >results/qrels.ohsu.88-91.aligned.txt
```

21. Use "analyze_results" to report recall, precision, f-measure, and cpu-time metrics for LSI and querytracer results.

```
$ cd results
$ analyze_results.exe query_matches_timing.txt qrels.ohsu.88-91.aligned.txt \
    >qt_results.txt
$ analyze_results.exe lsi_query_matches_timing.txt qrels.ohsu.88-91.aligned.txt \
    > lsi_results.txt
```

APPENDIX C

RESULTS – AVERAGE MEASUREMENTS

APPENDIX C

RESULTS – AVERAGE MEASUREMENTS

Average measurements of performance statistics from SCRIBE queries on four document collections.

Query	Relevant	Recall	Precision	F-measure	CPU time (sec.)
1	1	0.5239315	0.00525	0.010393628	0.6830435
2	1	0.022727275	0.00025	0.00049456	0.63823975
3	1	0.09012735	0.00325	0.00627322	0.69115075
4	1	0.56666675	0.00175	0.003487553	0.6363015
5	1	0.66883125	0.006	0.011888113	0.64629275
6	1	0.21666675	0.00125	0.002485583	0.63420575
7	1	0	0	0	0.66410475
8	0	0	0	0	0.5838175
9	1	0	0	0	0.6339295
10	1	0.475	0.00175	0.003486555	0.617034
11	1	0.9281045	0.01625	0.03194035	0.5923275
12	0	0	0	0	0.74102875
13	1	0.218589675	0.00275	0.00542751	0.7362995
14	1	0	0	0	0.667223
15	1	0.47970775	0.004	0.007931613	0.719395
16	1	0.41785725	0.004	0.00791695	0.64938525
17	1	0.4125	0.00225	0.004474155	0.70249025
18	1	0.352548	0.00675	0.013245428	0.73912725
19	1	0.070707175	0.0015	0.002937343	0.6193185
20	1	0.28214275	0.0015	0.0029831	0.642019
21	1	0.290673	0.008	0.015566225	0.6802515
22	1	0.1875	0.0005	0.00099701	0.75332275
23	1	0.237590775	0.00325	0.006409805	0.7028215
24	1	0.3732685	0.005	0.009854838	0.69455075
25	1	0.2947915	0.00325	0.00642397	0.706097
26	1	0.560065	0.00425	0.008432573	0.746392
27	0	0	0	0	0.673934
28	1	0.33049425	0.00375	0.007411103	0.48577425
29	1	0.03125	0.00025	0.000496033	0.59971775
30	1	0.2260235	0.008	0.01543403	0.7614705
31	1	0.217262	0.00175	0.00347027	0.288493
32	1	0.05792125	0.00075	0.00148075	0.707794
33	1	0.6397625	0.02175	0.0420633	0.69686925
34	0	0	0	0	0.76211875
35	1	0	0	0	0.71922125

Total Relevance Judgments	2637
Average Relevance Judgments	10.464

A zero (0) in the Relevant column means at least one sampled document collection contains no judged documents for the associated query.

Query	Relevant	Recall	Precision	F-measure	CPU time (sec.)
36	1	0.42140125	0.004	0.007923258	0.63108125
37	1	0.0516194	0.00075	0.001477845	0.6472265
38	1	0.39368675	0.00575	0.01133059	0.71165475
39	1	0.37365625	0.0105	0.020419175	0.71008675
40	1	0.25400425	0.003	0.00592887	0.75707275
41	1	0.14166675	0.00075	0.001492043	0.650754
42	1	0.11212125	0.00125	0.002472313	0.56678275
43	1	0	0	0	0.67756825
44	1	0.44496325	0.005	0.009881013	0.67425075
45	1	0.22291675	0.00125	0.002485588	0.7887275
46	1	0	0	0	0.52406475
47	1	0.072727275	0.0005	0.000992073	0.61156375
48	1	0.020833325	0.00025	0.00049407	0.38521975
49	1	0.677097	0.01175	0.023078975	0.71501725
50	1	0.08333325	0.00025	0.000498505	0.6967675
51	1	0.9613095	0.0125	0.02466955	0.7089535
52	1	0.33333325	0.00075	0.00149651	0.70277625
53	0	0	0	0	0.6300695
54	1	0.1666665	0.0005	0.00099701	0.62068725
55	1	0.64599575	0.00475	0.009424663	0.6447665
56	1	0.5	0.0025	0.00496825	0.7041675
57	1	0.56666675	0.00225	0.004473183	0.71497075
58	0	0	0	0	0.61711175
59	1	0	0	0	0.77412475
60	1	0.0694445	0.0005	0.000992558	0.74386875
61	1	0.261616025	0.00175	0.0034747	0.63434025
62	1	0.27083325	0.00125	0.002488058	0.68232425
63	1	0.048188025	0.00075	0.001476865	0.5973265

Average measurements of performance statistics from LSI queries on four document collections.

Query	Relevant	Recall	Precision	F-measure	CPU time
1	1	0.58739325	0.006	0.011875835	5.2684025
2	1	0.022727275	0.00025	0.00049456	9.28204
3	1	0.13069765	0.00475	0.009166188	8.3556525
4	1	0.56666675	0.00175	0.003487553	8.4680425
5	1	0.84383125	0.007	0.0138757	7.5338525
6	1	0.21666675	0.00125	0.002485583	9.9270025
7	1	0	0	0	8.091595
8	0	0	0	0	8.7086
9	1	0	0	0	8.807995
10	1	0.475	0.00175	0.003486555	15.379985
11	1	1	0.0175	0.0343976	13.3731725
12	0	0	0	0	7.717355
13	1	0.4128205	0.005	0.009871268	8.7292025
14	1	0	0	0	7.7111425
15	1	0.86931825	0.00725	0.0143766	8.32133
16	1	0.4357145	0.00425	0.00841005	7.5168475
17	1	0.9	0.00525	0.01043492	7.857525
18	1	0.37907275	0.00725	0.014226795	10.7330775
19	1	0.10656555	0.002	0.003924485	8.9549925
20	1	0.54285725	0.00275	0.005470668	9.0159
21	1	0.330673	0.009	0.01751745	7.5927725
22	1	0.1875	0.0005	0.00099701	8.337425
23	1	0.41624825	0.006	0.011825615	12.69912
24	1	0.40411225	0.00575	0.011322535	8.4784675
25	1	0.534028	0.006	0.011856848	7.8075725
26	1	0.641234	0.005	0.009918223	11.477455
27	0	0	0	0	8.502355
28	1	0.18489	0.002	0.003955033	11.06997
29	1	0.03125	0.00025	0.000496033	9.3821325
30	1	0.233353	0.00825	0.01591308	11.09067
31	1	0.86011925	0.00725	0.014370248	8.7855825
32	1	0.08292125	0.001	0.0019758	8.7793525
33	1	0.871406	0.02975	0.057526525	8.835965
34	0	0	0	0	8.437975
35	1	0.010416675	0.00025	0.000488283	8.189445
36	1	0.4633835	0.0045	0.008911888	10.9852575
37	1	0.1016194	0.0015	0.002955678	8.872905
38	1	0.3681815	0.0055	0.01083456	9.2548525
39	1	0.38758275	0.01075	0.020912225	7.6124225

Total Relevance Judgments 2637

Average Relevance Judgments 10.464 29

A zero (0) in the Relevant column means at least one sampled document collection contains no judged documents for the associated query.

Query	Relevant	Recall	Precision	F-measure	CPU time
40	1	0.2998375	0.0035	0.006917988	7.875615
41	1	0.14166675	0.00075	0.001492043	11.1742025
42	1	0.1579545	0.00175	0.003461433	9.15559
43	1	0	0	0	8.58951
44	1	0.49175825	0.00575	0.011358345	9.935005
45	1	0.29583325	0.00175	0.003478635	7.2266875
46	1	0.08333325	0.00025	0.000498505	10.6742925
47	1	0.122727275	0.00075	0.001489585	10.01882
48	1	0.038690475	0.0005	0.000987168	10.4616225
49	1	0.971875	0.0165	0.0324198	15.48941
50	1	0.08333325	0.00025	0.000498505	7.152795
51	1	0.97916675	0.01275	0.02516265	8.42641
52	1	0.5	0.001	0.00199601	9.7805175
53	0	0	0	0	10.0455275
54	1	0.25	0.00075	0.001495515	9.0622575
55	1	0.718723	0.00525	0.010416725	9.530765
56	1	0.71666675	0.0035	0.00695536	9.88718
57	1	0.56666675	0.00225	0.004473183	9.75889
58	0	0	0	0	8.789675
59	1	0.1704545	0.0015	0.002971783	10.0661575
60	1	0.0694445	0.0005	0.000992558	8.616985
61	1	0.39924225	0.003	0.005950938	9.3699325
62	1	0.5	0.00225	0.004479103	8.3847075
63	1	0.02777775	0.00025	0.00049554	10.47364

APPENDIX D

ABRIDGED MANN-WHITNEY U TESTS

APPENDIX D

ABRIDGED MANN-WHITNEY U TESTS

Mann-Whitney rank sum test over Recall metric

Source	Recall	Rank
qt	0	1
qt	0	2
qt	0	3
qt	0	4
qt	0	5
qt	0	6
qt	0	7
lsi	0	8
lsi	0	9
lsi	0	10
lsi	0	11
lsi	0.010417	12
qt	0.020833	13
qt	0.022727	14
lsi	0.022727	15
lsi	0.027778	16
qt	0.03125	17
lsi	0.03125	18
lsi	0.03869	19
qt	0.048188	20
qt	0.051619	21
qt	0.057921	22
qt	0.069445	23
lsi	0.069445	24
qt	0.070707	25
qt	0.072727	26

Source	Rank Sum	Samples
qt	3000	57

Source	Rank Sum	Samples
lsi	3555	57

Mann-Whitney U Statistic

1902

? Statistic

0.585411

z - Normal Distribution

1.572642

Mann-Whitney rank sum test over Precision metric

Source	Precision	Rank
qt	0	1
qt	0	2
qt	0	3
qt	0	4
qt	0	5
qt	0	6
qt	0	7
lsi	0	8
lsi	0	9
lsi	0	10
lsi	0	11
qt	0.00025	12
qt	0.00025	13
qt	0.00025	14
qt	0.00025	15
lsi	0.00025	16
lsi	0.00025	17
lsi	0.00025	18
lsi	0.00025	19
lsi	0.00025	20
lsi	0.00025	21
qt	0.0005	22
qt	0.0005	23
qt	0.0005	24
qt	0.0005	25
lsi	0.0005	26

Source	Rank Sum	Samples
qt	3015	57

Source	Rank Sum	Samples
lsi	3540	57

Mann-Whitney U Statistic

1887

? Statistic

0.580794

z - Normal Distribution

1.487634

Mann-Whitney rank sum test over F-measure metric

Source	F-measure	Rank
qt	0	1
qt	0	2
qt	0	3
qt	0	4
qt	0	5
qt	0	6
qt	0	7
lsi	0	8
lsi	0	9
lsi	0	10
lsi	0	11
lsi	0.000488	12
qt	0.000494	13
qt	0.000495	14
lsi	0.000495	15
lsi	0.000496	16
qt	0.000496	17
lsi	0.000496	18
qt	0.000499	19
lsi	0.000499	20
lsi	0.000499	21
lsi	0.000987	22
qt	0.000992	23
qt	0.000993	24
lsi	0.000993	25
qt	0.000997	26

Source	Rank Sum	Samples
qt	3049	57

Source	Rank Sum	Samples
lsi	3506	57

Mann-Whitney U Statistic

1853

? Statistic

0.570329

z - Normal Distribution

1.29495

Mann-Whitney rank sum test over CPU Time metric

Source	CPU Time	Rank
qt	0.288493	1
qt	0.38522	2
qt	0.485774	3
qt	0.524065	4
qt	0.566783	5
qt	0.592328	6
qt	0.597327	7
qt	0.599718	8
qt	0.611564	9
qt	0.617034	10
qt	0.619319	11
qt	0.620687	12
qt	0.631081	13
qt	0.63393	14
qt	0.634206	15
qt	0.63434	16
qt	0.636302	17
qt	0.63824	18
qt	0.642019	19
qt	0.644767	20
qt	0.646293	21
qt	0.647227	22
qt	0.649385	23
qt	0.650754	24
qt	0.664105	25
qt	0.667223	26

Source	Rank Sum	Samples
qt	1653	57

Source	Rank Sum	Samples
lsi	4902	57

Mann-Whitney U Statistic
3249

? Statistic
1

z - Normal Distribution
9.206331

APPENDIX E

ANOVA POWER ANALYSIS

APPENDIX E

ANOVA POWER ANALYSIS

SAS ANOVA power analysis program

```
options nodate nocenter nonumber;
title 'Power analysis for ANOVA designs';
%include "fpower.sas";
%fpower( a=2, b=1, delta=%str(0.2, 0.5, 0.8, 1.0, 2.0,
3.0),
         alpha=0.05, n=%str(2 to 10 by 1, 12 to 18 by 2,
                            20 to 40 by 10, 50,
                            60 to 70 by 1),
         ptable=YES, ntable=NO);
```

SAS power table output

Power analysis for ANOVA designs
2x 1 layout Ha: T1=GM-Delta/2, T2=T3=...=T(k-1)=GM, Tk=GM+Delta/2 tested at Alpha= 0.050

DELTA (in units of sigma=Std. Dev.)

N	0.200	0.500	0.800	1.000	2.000	3.000
2	0.051	0.061	0.079	0.095	0.218	0.387
3	0.054	0.076	0.119	0.158	0.462	0.782
4	0.056	0.092	0.160	0.223	0.656	0.938
5	0.059	0.107	0.200	0.286	0.790	0.984
6	0.061	0.123	0.241	0.347	0.876	0.996
7	0.063	0.138	0.280	0.405	0.929	0.999
8	0.066	0.154	0.319	0.461	0.960	0.999
9	0.068	0.169	0.357	0.513	0.978	0.999
10	0.070	0.185	0.395	0.562	0.988	0.999
12	0.075	0.216	0.465	0.648	0.996	0.999
14	0.080	0.247	0.531	0.721	0.999	0.999
16	0.085	0.277	0.590	0.781	0.999	0.999
18	0.089	0.308	0.645	0.830	0.999	1.000
20	0.094	0.337	0.693	0.868	0.999	1.000
30	0.118	0.477	0.861	0.967	0.999	1.000
40	0.143	0.598	0.942	0.992	1.000	1.000
50	0.167	0.696	0.977	0.998	1.000	1.000
60	0.192	0.775	0.991	0.999	1.000	1.000
61	0.194	0.782	0.992	0.999	1.000	1.000
62	0.197	0.788	0.993	0.999	1.000	1.000
63	0.199	0.795	0.993	0.999	1.000	1.000
64	0.202	0.801	0.994	0.999	1.000	1.000
65	0.204	0.807	0.994	0.999	1.000	1.000
66	0.207	0.813	0.995	0.999	1.000	1.000
67	0.209	0.819	0.995	0.999	1.000	1.000
68	0.212	0.824	0.996	0.999	1.000	1.000
69	0.214	0.830	0.996	0.999	1.000	1.000
70	0.217	0.835	0.996	0.999	1.000	1.000

APPENDIX F

ANOVA RESULTS

Oneway

Descriptives

		N	Mean	Std. Deviation	Std. Error	95% Confidence Interval for Mean		Minimum	Maximum
						Lower Bound	Upper Bound		
Recall	1	63	.25820302	.242476822	.030549208	.19713603	.31927002	.000000	.961310
	2	63	.33624448	.303505215	.038238063	.25980768	.41268128	.000000	1.000000
	Total	126	.29722375	.276380840	.024621962	.24849383	.34595367	.000000	1.000000
Precision	1	63	.00303968	.004162333	.000524405	.00199141	.00408795	.000000	.021750
	2	63	.00384127	.005112505	.000644115	.00255370	.00512884	.000000	.029750
	Total	126	.00344048	.004660411	.000415182	.00261878	.00426217	.000000	.029750
F_measure	1	63	.00597367	.008120881	.001023135	.00392845	.00801889	.000000	.042063
	2	63	.00755163	.009965103	.001255485	.00504195	.01006131	.000000	.057527
	Total	126	.00676265	.009088039	.000809627	.00516030	.00836500	.000000	.057527
CPU_time	1	63	.66252	.084707	.010672	.64119	.68386	.288	.789
	2	63	9.29987	1.762509	.222055	8.85598	9.74375	5.268	15.489
	Total	126	4.98119	4.510487	.401826	4.18593	5.77646	.288	15.489

ANOVA

		Sum of Squares	df	Mean Square	F	Sig.
Recall	Between Groups	.192	1	.192	2.543	.113
	Within Groups	9.356	124	.075		
	Total	9.548	125			
Precision	Between Groups	.000	1	.000	.931	.336
	Within Groups	.003	124	.000		
	Total	.003	125			
F_measure	Between Groups	.000	1	.000	.949	.332
	Within Groups	.010	124	.000		
	Total	.010	125			
CPU_time	Between Groups	2350.017	1	2350.017	1509.511	.000
	Within Groups	193.044	124	1.557		
	Total	2543.061	125			