Accepted for the *Grace Hopper Celebration of Women in Computing 2004*

# Assessing the Impact of Sparsification on LSI Performance

April Kontostathis
Department of Mathematics and Computer Science
Ursinus College
Collegeville PA 19426
akontostathis@ursinus.edu

William M. Pottenger and Brian D. Davison
Department of Computer Science and Engineering
Lehigh University
Bethlehem PA 18015
billp,davison@cse.lehigh.edu

## ABSTRACT

We describe an approach to information retrieval using Latent Semantic Indexing (LSI) that directly manipulates the values in the Singular Value Decomposition (SVD) matrices. We convert the dense term by dimension matrix into a sparse matrix by removing a fixed percentage of the values. We present retrieval and runtime performance results, using seven collections, which show that using this technique to remove up 70% of the values in the term by dimension matrix results in similar or improved retrieval performance (as compared to LSI), while reducing memory requirements and query response time. Removal of 90% of the values results in significantly reduced memory requirements and dramatic improvements in query response time. Removal of 90% of the values degrades retrieval performance slightly for smaller collections, but improves retrieval performance by 60% on the large collection we tested.

## 1 INTRODUCTION

The amount of textual information available in digital formats is continuing to grow rapidly. It is impossible for a single individual to read and understand all of the literature that is available for any given topic. Researchers in information retrieval, computational linguistics and textual data mining are working on the development of methods to process this data and present it in a usable format.

Many algorithms for searching textual collections have been developed, and, in this paper, we focus our study on one system, Latent Semantic Indexing (LSI). LSI was developed in the early 1990s and has been applied to a wide variety of learning tasks that involve textual data [4, 6, 14, 12]. LSI is based upon a linear algebraic technique for factoring matrices called Singular Value Decomposition (SVD). In LSI, the dimensionally of the text retrieval problem is reduced by truncating the matrices produced by the SVD process.

Several outstanding issues remain with LSI. First, the SVD algorithm is computationally expensive. Many approximation algorithms have been implemented to resolve this problem [1]. Second, choosing an optimal dimensionality reduction parameter ($k$) for each collection remains elusive. Traditionally, the optimal $k$ has been chosen by running a set of queries with known relevant document sets against the SVD matrices for multiple values of $k$. The $k$ that results in the best retrieval performance is chosen as the optimal $k$ for each collection. Optimal $k$ values are typically in the range of 100-300 dimensions [6]. Unfortunately, a $k$ chosen using this technique is only optimized to the queries in the training set.

The third issue that remains with the use of LSI is the density of the matrices after decomposition. The original term by document matrix is very sparse, but the $T$ and $D$ matrices are dense, having few zero values. This density results in an increase in memory requirements for the LSI system, and an increase in computation cost when running the queries.

This paper addresses this third point. We present an algorithm that removes (zeroes out) a large portion of the values in the $T_k$ and $S_kD_k$ matrices. In Section 3 we describe the LSI processing steps. Section 4 describes our approach to reducing the density of the SVD matrices in detail. In Section 5 we show the impact of this approach on retrieval quality and runtime performance.

## 2 RELATED WORK

While others have reportedly built large-scale systems utilizing LSI (e.g., the Excite search engine [13]), to our knowledge, only Chen et al. [3] have developed and published a description of a comprehensive system which directly addresses the implementation issues required in an LSI system. The authors describe two approaches for reducing memory requirements and improving run time efficiency during the query processing phase of LSI. The first method involves compressing each entry of the vector to an 8-bit char instead of a 4-byte float. A lookup table is also implemented to reduce the number of multiplications needed when comparing the query vector to the document vectors. The second approach involves the development of document clusters, which are then selected and compared to the query vector. This approach vastly reduces the number of documents which need to be compared, and, thus, the number of computations. The authors do not describe their clustering algorithm in detail.

Another paper by Gao and Zhang [9] discusses an alternative approach to applying sparsification to the LSI term by dimension and document dimension matrices. The work by Gao and Zhang was developed independently, and used only three small collections for evaluation. Our approach to sparsification is different from the ones used in [9]; furthermore, we present data for seven collections and also collected data related to the run time considerations, as well as the retrieval effectiveness, of sparsification.

# 3 BACKGROUND

In this section we provide background information on the necessary processing required in a retrieval system that uses LSI. We also discuss our approach to evaluation.

## 3.1 Latent Semantic Indexing

Latent Semantic Indexing (LSI) [4] is a well-known technique used in information retrieval. LSI has been applied to a wide variety of learning tasks, such as search and retrieval [4, 6], classification [14] and filtering [6, 7]. LSI is a vector space approach for modeling documents, and many have claimed that the technique brings out the 'latent' semantics in a collection of documents [4, 6].

LSI is based on a mathematical technique called Singular Value Decomposition (SVD). The SVD process decomposes a term by document matrix into three matrices: a term by dimension matrix, $T$, a singular value matrix, $S$, and a document by dimension matrix, $D$. The number of dimensions is min $(t, d)$ where $t$ = number of terms and $d$ = number of documents. The original matrix can be obtained, through matrix multiplication of $TSD^T$.

In the LSI system, the $T$, $S$ and $D$ matrices are truncated to $k$ dimensions. Dimensionality reduction reduces "noise" in the term–document matrix resulting in a richer word relationship structure that many researchers claim reveals latent semantics present in the collection [4, 6]. Queries are represented in the reduced space by $T_k^T q$, where $T_k^T$ is the transpose of the term by dimension matrix, after truncation to $k$ dimensions. Queries are compared to the reduced document vectors, scaled by the singular values ($S_k D_k$), by computing the cosine similarity. This process provides a mechanism to rank the document set for each query.

## 3.2 Evaluation

Retrieval quality for an information retrieval system can be expressed in a variety of ways. In the current work, we use precision and recall to express the quality of an information retrieval system. Precision is defined as the percentage of retrieved documents which are relevant to the query. Recall is the percentage of all relevant documents that were retrieved.

These metrics can be applied in two ways. First, we can compute recall and precision at rank = $n$, where $n$ is a constant. In this case, we look at the first $n$ documents returned from the query and compute the precision and recall using the above definitions. An alternative approach involves computing precision at a given recall level. In this second case, we continue to retrieve documents until a given percentage of correct documents has been retrieved (for example, 25%), and then compute the precision. In Section 5, we apply this second approach to evaluate of our sparsification strategy.

Precision and recall require the existence of collections that contain a group of documents, a set of standard queries and a set of relevance judgments (a list of which documents are relevant to which query, and which are not relevant). We used seven such collections during the course of our study. The collections we used are summarized in Table 1. These collections were downloaded from a variety of sources. MED, CISI, CRAN, NPL, and CACM were downloaded from the SMART web site at Cornell University. LISA was obtained from the Information Retrieval Group web site at the University of Glasgow. The OHSUMED collection was downloaded from the Text Retrieval Conference (TREC) web site at the National Institute of Standards and Technology. Not all of the documents in the OHSUMED collection have been judged for relevance for each query. In our experiments, we calculated precision and recall by assuming that all unjudged documents are not relevant. Similar studies that calculate precision using only the judged documents are left to future work.

# 4 SPARSIFICATION OF THE LSI MATRICES

This paper reports the results of study to determine the most critical elements of the $T_k$ and $S_k D_k$ matrices, which are input to LSI. We are interested in the impact, both in terms of retrieval quality and query run time performance, of the removal of a large portion of the entries in these matrices. Several patterns were identified in our preliminary work. For example, removal of all negative elements severely degrades performance, as does removal of 'too many' of the $S_k D_k$ elements. However, a large portion of the $T_k$ values can be removed without a significant change in retrieval performance.

In this section we describe the algorithm we used to remove values from the $T_k$ and $S_k D_k$ matrices. In Section 5, we describe the impact of this sparsification strategy on retrieval quality and query run time performance.

## 4.1 Methodology

Our sparsification algorithm focuses on the values with absolute value near zero, and we ask the question: "How many values can we remove without severely impacting retrieval performance?" Intuitively, the elements of the row vectors in the $T_k S_k$ matrix and the column vectors in the $S_k D_k$ matrix can be used to describe the importance of each term (document) along a given dimension. This approach stems from the premise that each dimension in the SVD represents a given concept or group of concepts [12]. Positive values indicate a similarity between the term (document) and the concept; negative values indicate dissimilarity. Because of

Table 1: Collections used to compare Sparsification Strategy to Traditional LSI

| **Identifier** | **Description** | **Docs** | **Terms** | **Queries** |
|---|---|---|---|---|
| MED | Medical abstracts | 1033 | 5831 | 30 |
| CISI | Information science abstracts | 1450 | 5143 | 76 |
| CACM | Communications of the ACM abstracts | 3204 | 4863 | 52 |
| CRAN | Cranfield collection | 1400 | 3932 | 225 |
| LISA | Library and Information Science Abstracts | 6004 | 18429 | 35 |
| NPL | Larger collection of very short documents | 11429 | 6988 | 93 |
| OHSUMED | Clinically-oriented MEDLINE subset | 348566 | 170347 | 106 |

Compute $T_kS_k$ and $S_kD_k$
Determine $PosThres$: The threshold that would result in removal of $x\%$ of the positive elements of $T_kS_k$
Determine $NegThres$: The threshold that would result in removal of $x\%$ of the negative elements of $T_kS_k$
For each element of $T_k$, change to zero, if the corresponding element of $T_kS_k$ Falls between $PosThres$ and $NegThres$
For each element of $S_kD_k$, change to zero, if it falls between $PosThres$ and $NegThres$

Figure 1: Sparsification Algorithm

this association between the $T_kS_k$ and $S_kD_k$ matrix values, we decided that the best approach would define a common truncation value, based on the values in the $T_kS_k$ matrix, which would be used for both the $T_k$ and $S_kD_k$ matrices. Furthermore, since the negative values are important, we wanted to retain an equal number of positive and negative values in the $T_k$ matrix. The algorithm we used is outlined in Figure 1. We chose positive and negative threshold values that are based on the $T_kS_k$ matrix and that result in the removal of a fixed percentage of the $T_k$ matrix. We use these values to truncate both the $T_k$ and $S_kD_k$ matrices.

Our approach was tested using seven collections. These collections have relevance judgments available for each query and have been used by other researchers to test the performance of search and retrieval algorithms [4, 10, 8, 11]. These collections are summarized in Table 1.

The Parallel General Text Parser (PGTP) [2] was used to preprocess the text data, including creation and decomposition of the term document matrix. For our experiments, we applied the log entropy weighting option and normalized the document vectors. The sparsification algorithm was applied to each of our collections, using truncation percentages of 10% to 90%.

Retrieval quality and query runtime performance measurements were taken at multiple values of $k$. The values of $k$ for the smaller collections ranged from 25 to 200; $k$ values from 50 to 500 were used for testing the larger collections. These values of $k$ were chosen because smaller $k$ values are preferred when using LSI, due to the computational cost associated with the SVD algorithm, as well as the computation cost of storing and comparing large dimension vectors.

# 5 RESULTS

In this section we describe our experimental results. We show that our approach substantially reduces the RAM required to run an LSI system without compromising retrieval quality. We also show that our approach provides a significant improvement in query run time efficiency.

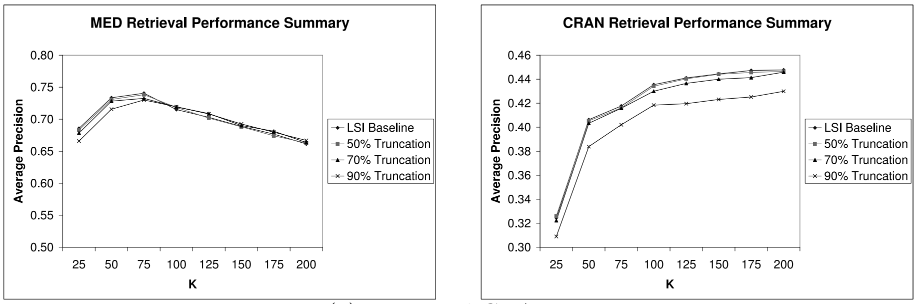
## 5.1 Impact on Retrieval Quality

The retrieval quality results for three different truncation values for the collections studied are shown in Figure 2. Retrieval quality for our sparsified LSI was compared to a standard LSI system. The average precision values for both the baseline and for the experimental technique were obtained by computing the precision of each query at recall levels .25, .50, and .75.

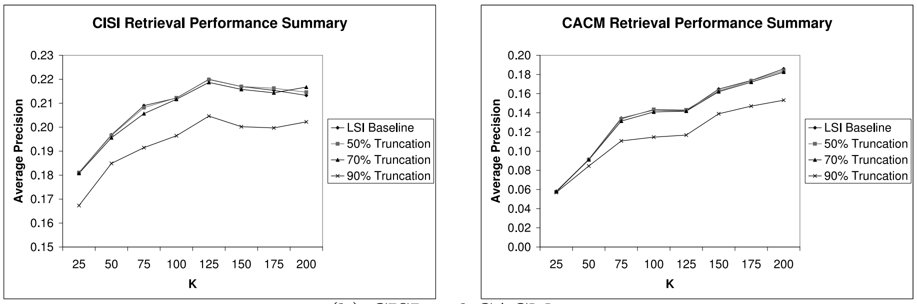
Figure 2 demonstrates that removal of 50% of the $T_k$ matrix values resulted in retrieval quality that is indistinguishable from the LSI baseline for all seven collections we tested. In most cases, sparsification up to 70% can be achieved, particularly at better performing values of $k$, without a significant impact on retrieval quality. For example, $k$=500 for NPL and $k$=200 for CISI have performance near or greater than the LSI baseline when 70% of the values are removed.

## 5.2 Impact on Query Run Time Performance

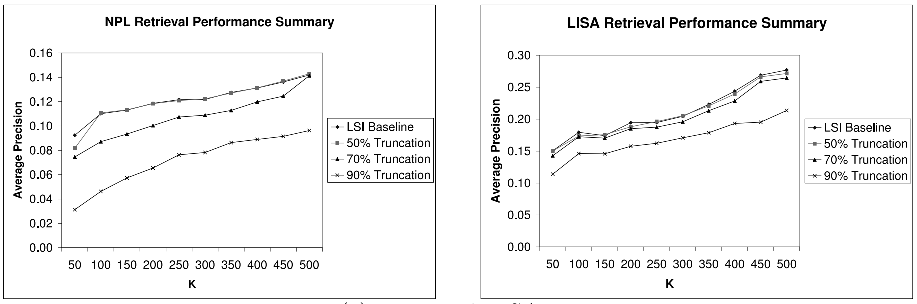
In order to determine the impact of sparsification on query run time performance we implemented a sparse matrix version of the LSI query processing. The well-known compressed row storage format for sparse matrices was used to



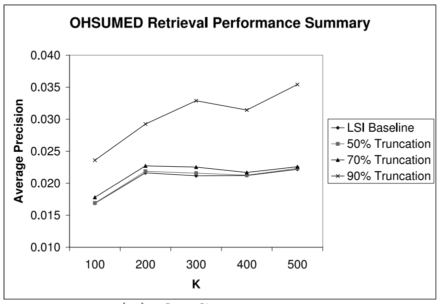
(a) MED and CRAN



(b) CISI and CACM



(c) NPL and LISA



(d) OHSUMED

Figure 2: Retrieval Quality of Sparsified LSI vs. LSI

Table 2: RAM Savings for T and D matrices

| Collection | $k$ | Sparsification Level | Sparsified RAM (MB) | LSI RAM (MB) | Improvement (%) |
|---|---|---|---|---|---|
| MED | 75 | 70 | 3.7 | 7.9 | 53 |
| CISI | 125 | 70 | 6.3 | 12.6 | 50 |
| CRAN | 200 | 70 | 8.3 | 16.3 | 49 |
| CACM | 200 | 70 | 14.4 | 24.6 | 42 |
| NPL | 500 | 70 | 93.1 | 140.5 | 34 |
| LISA | 500 | 70 | 66.8 | 115.7 | 42 |
| OHSUMED | 500 | 70 | 3217 | 3959 | 19 |
| MED | 75 | 90 | 1.6 | 7.9 | 79 |
| CISI | 125 | 90 | 2.9 | 12.6 | 77 |
| CRAN | 200 | 90 | 3.6 | 16.3 | 78 |
| CACM | 200 | 90 | 6.3 | 24.6 | 75 |
| NPL | 500 | 90 | 29.0 | 140.5 | 79 |
| LISA | 500 | 90 | 28.3 | 115.7 | 76 |
| OHSUMED | 500 | 90 | 2051 | 3959 | 48 |

store the new sparse matrices generated by our algorithm [5].

In the compressed row storage format, the nonzero elements of a matrix are stored as a vector of values. Two additional vectors are used to identify the coordinates of each value: a row pointer vector identifies the position of the first nonzero element in each row, and a column indicator vector identifies the column corresponding to each element in the value vector. This storage format requires a vector of length num-nonzeroes to store the actual values, a vector of length num-nonzeroes to identify the column corresponding to each value, and a vector of length num-rows to identify the starting position of each row. Table 2 shows that this approach significantly reduces the RAM requirements of LSI. This reduction is due to our sparsification strategy, which produces relative sparse matrix. Implementation of the compressed row storage format for a non-sparsified LSI system would result in an increase in the memory requirements.

When comparing the runtime considerations of our approach to LSI, we acknowledge that our approach requires additional preprocessing, as we implement two additional steps, determining the threshold value and applying the threshold to the $T_k$ and $S_kD_k$ matrices. These steps are applied once per collection, however, and multiple queries can then be run against the collection.

The query processing for LSI is comprised of two primary tasks: development of the pseudo query, which relies on the $T_k$ matrix, and the comparison of the pseudo query to the documents, which uses the $S_kD_k$ matrix. Table 3 indicates that the $S_kD_k$ sparsification ranges from 18% to 33%, when 70% of the $\mathrm{T}_k$ values are removed. A much larger $S_kD_k$ sparsification range of 43%-80% is achieved at a 90% reduction in the $T_k$ matrix.

The number of cpu cycles required to run all queries in each collection was collected using the clock() function available in C++. Measurements were taken for both the baseline LSI code and the sparsified code. Each collection was tested twice, and the results in Table 3 represent the average of the two runs for selected records.

Sparsification of the matrix elements results in an improvement in query runtime performance for all collections, with the exception of MED and NPL at 70% sparsification. The data implies that, for most collections, query run time performance improves as the number of entries in the document vectors is reduced. Figure 2 shows a degradation in retrieval performance at 90% sparsification for the smaller collections; however, OHSUMED retrieval quality improves dramatically at 90% sparsification.

# 6 CONCLUSIONS

We conclude from this data that query run time improvements in LSI can be achieved using our sparsification strategy for many collections. Our approach zeroes a fixed percentage of both positive and negative values of the term and document vectors produced by the SVD process. Our data shows that, for small collections, we can successfully reduce the RAM requirements by 45% (on average), and the query response time an average of 3%, without sacrificing retrieval quality. If a slight degradation in retrieval quality is acceptable, the RAM requirements can be reduced by 77%, and query run time can be reduced by 40% for smaller collections using our approach.

On the larger TREC collection (OHSUMED), we can reduce the runtime by 30%, reduce the memory required by 48% and improve retrieval quality by 60% by implementing our sparsification algorithm at 90%.

## Acknowledgements

This work was supported in part by National Science Foundation Grant Number EIA-0087977 and the National Computational Science Alliance under IRI030006 (we utilized the IBM pSeries 690 cluster). The authors appreciate the assistance provided by their colleagues at Lehigh University and Ursinus College. Co-author William M. Pottenger also gratefully acknowledges his Lord and Savior, Yeshua (Jesus) the Messiah, for His continuing guidance in and salvation of his life.

Table 3: Percentage of Document Vector Entries Removed

| Collection | $k$ | Term Spars (%) | Doc Spars (%) | Run Time Improvement (%) |
|---|---|---|---|---|
| MED | 75 | 70 | 23 | -1 |
| CISI | 125 | 70 | 26 | 1 |
| CRAN | 200 | 70 | 29 | 6 |
| CACM | 200 | 70 | 28 | 3 |
| NPL | 500 | 70 | 33 | -3 |
| LISA | 500 | 70 | 29 | 10 |
| OHSUMED | 500 | 70 | 18 | 3 |
| MED | 75 | 90 | 47 | 16 |
| CISI | 125 | 90 | 53 | 27 |
| CRAN | 200 | 90 | 61 | 37 |
| CACM | 200 | 90 | 64 | 40 |
| NPL | 500 | 90 | 80 | 66 |
| LISA | 500 | 90 | 66 | 54 |
| OHSUMED | 500 | 90 | 43 | 30 |

# REFERENCES


[1] Michael W. Berry, Teresa Do, Gavin O'Brien, Vijay Krishna, and Sowmini Varadhan. SVDPACKC (version 1.0) User's Guide. Technical report, 1993.

[2] Michael W. Berry and Dian I. Martin. Principal component analysis for information retrieval. In E. J. Kontoghiorghes, editor, *Handbook of Parallel Computing and Statistics*. Marcel Dekker, New York, 2004. In press.

[3] Chung-Min Chen, Ned Stoffel, Mike Post, Chumki Basu, Devasis Bassu, and Clifford Behrens. Telcordia LSI engine: Implementation and scalability issues. In *Proceedings of the Eleventh International Workshop on Research Issues in Data Engineering (RIDE 2001)*, Heidelberg, Germany, April 2001.

[4] Scott C. Deerwester, Susan T. Dumais, Thomas K. Landauer, George W. Furnas, and Richard A. Harshman. Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, 41(6):391–407, 1990.

[5] Jack Dongarra. Sparse matrix storage formats. In Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst, editors, *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide*, pages 372–378. SIAM, Philadelphia, 2000.

[6] Susan T. Dumais. LSI meets TREC: A status report. In D. Harman, editor, *The First Text REtrieval Conference (TREC-1), National Institute of Standards and Technology Special Publication 500-207*, pages 137–152, 1992.

[7] Susan T. Dumais. Latent semantic indexing (LSI) and TREC-2. In D. Harman, editor, *The Second Text REtrieval Conference (TREC-2), National Institute of Standards and Technology Special Publication 500-215*, pages 105–116, 1994.

[8] Abdelmoula El-Hamdouchi and Peter Willett. Hierarchic document clustering using Ward's method. In *Proceedings of the Ninth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 149–156, 1986.

[9] Jing Gao and Jun Zhang. Sparsification strategies in latent semantic indexing. In M. W. Berry and W. M. Pottenger, editors, *Proceedings of the 2003 Text Mining Workshop*, May 2003.

[10] George Karypis and Eui-Hong Han. Fast supervised dimensionality reduction algorithm with applications to document categorization & retrieval. In *Proceedings of CIKM 2000*, pages 12–19, 2000.

[11] Stefan Klink, Armin Hust, Markus Junker, and Andreas Dengel. Improving document retrieval by automatic query expansion using collaborative learning of term-based concepts. In *Proceedings of the 5th International Workshop on Document Analysis Systems (DAS)*, volume 2423 of *Lecture Notes in Computer Science*, pages 376–387, Princeton, NJ, USA, August 2002. Springer.

[12] Hinrich Schütze. Dimensions of meaning. In *Proceedings of Supercomputing*, pages 787–796, 1992.

[13] John D. Zakis and Zenon J. Pudlowski. The world wide web as universal medium for scholarly publication, information retrieval and interchange. *Global Journal of Engineering Education*, 1(3), 1997.

[14] Sarah Zelikovitz and Haym Hirsh. Using LSI for text classification in the presence of background text. In H. Paques, L. Liu, and D. Grossman, editors, *Proceedings of CIKM-01, tenth ACM International Conference on Information and Knowledge Management*, pages 113–118, Atlanta, GA, 2001. ACM Press, New York.