Chapter in T. Y. Lin, S. Ohsuga, C.-J. Liau, X. T. Hu, and S. Tsumoto (eds.), *Foundations of Data Mining and Knowledge Discovery*, Springer-Verlag, 2005.

Identification of Critical Values in Latent Semantic Indexing

April Kontostathis[1], William M. Pottenger[2], and Brian D. Davison[2]

[1] Ursinus College, Department of Mathematics and Computer Science
P.O. Box 1000 (601 Main St.), Collegeville, PA 19426
akontostathis@ursinus.edu

[2] Lehigh University, Department of Computer Science and Engineering
19 Memorial Drive West, Bethlehem, PA 18015
billp,davison@cse.lehigh.edu

In this chapter we analyze the values used by Latent Sematic Indexing (LSI) for information retrieval. By manipulating the values in the Singular Value Decomposition (SVD) matrices, we find that a significant fraction of the values have little effect on overall performance, and can thus be removed (changed to zero). This allows us to convert the dense term by dimension and document by dimension matrices into sparse matrices by identifying and removing those entries. We empirically show that these entries are unimportant by presenting retrieval and runtime performance results, using seven collections, which show that removal of up 70% of the values in the term by dimension matrix results in similar or improved retrieval performance (as compared to LSI). Removal of 90% of the values degrades retrieval performance slightly for smaller collections, but improves retrieval performance by 60% on the large TREC collection we tested. Our approach additionally has the computational benefit of reducing memory requirements and query response time.

1 Introduction

The amount of textual information available digitally is overwhelming. 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 chapter we focus on one such system: Latent Semantic Indexing (LSI). LSI was developed in the early 1990s [5] and has been applied to a wide variety of tasks that involve textual data [5, 8, 19, 22, 9, 10]. LSI is based upon a linear algebraic technique for factoring matrices called Singular

Value Decomposition (SVD). In previous work [16, 17, 15] we noted a strong correlation between the distribution of term similarity values (defined as the cosine distance between the vectors in the term by dimension matrix) and the performance of LSI. In the current work, we continue our analysis of the values in the truncated term by dimension and document by dimension matrices. We describe a study to determine the most critical elements of these matrices, which are input to the query matching step in LSI. We hypothesize that identification and zeroing (removal) of the least important entries in these matrices will result in a more computationally efficient implementation, with little or no sacrifice in the retrieval effectiveness compared to a traditional LSI system.

2 Background and Related Work

Latent Semantic Indexing (LSI) [5] is a well-known technique used in information retrieval. LSI has been applied to a wide variety of tasks, such as search and retrieval [5, 8], classification [22] and filtering [9, 10]. 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 [5, 8].

LSI is based on a mathematical technique called Singular Value Decomposition (SVD) [11]. The SVD process decomposes a term by document matrix, A, 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 reveals latent semantics present in the collection. 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.

The algebraic foundation for Latent Semantic Indexing (LSI) was first described in [5] and has been further discussed by Berry, et al. in [1, 2]. These papers describe the SVD process and interpret the resulting matrices in a geometric context. They show that the SVD, truncated to k dimensions, gives the optimal rank-k approximation to the original matrix. Wiemer-Hastings shows that the power of LSI comes primarily from the SVD algorithm [21].

Other researchers have proposed theoretical approaches to understanding LSI. Zha and Simon describe LSI in terms of a subspace model and propose a statistical test for choosing the optimal number of dimensions for a given collection [23]. Story discusses LSI's relationship to statistical regression and Bayesian methods [20]. Ding constructs a statistical model for LSI using the cosine similarity measure, showing that the term similarity and document similarity matrices are formed during the maximum likelihood estimation, and LSI is the optimal solution to this model [6].

Although other researchers have explored the SVD algorithm to provide an understanding of SVD-based information retrieval systems, to our knowledge, only Schütze has studied the values produced by LSI [18]. We expand upon this work in [16, 17, 15], showing that SVD exploits higher order term co-occurrence in a collection, and showing the correlation between the values produced in the term-term matrix and the performance of LSI. In the current work, we extend these results to determine the most critical values in an LSI system.

Other researchers [4, 12] have recently applied sparsification techniques to reduce the computation cost of LSI. These papers provide further empirical evidence for our claim that the retrieval performance of LSI depends on a subset of the SVD matrix elements. Gao and Zhang have simultaneously, but independently, proposed mechanisms to take the dense lower-dimensional matrices that result from SVD truncation, and make them sparse [12]. Chen et al. implicitly do this as well, by encoding values into a small set of discrete values [4]. Our approach to sparsification (described in Section 3) is somewhat different from the ones used by Gao and Zhang; furthermore, we present data for additional collections (they used three small collections), and also describe the run time considerations, as well as the retrieval effectiveness, of sparsification.

3 Sparsification of the LSI Input Matrices

This paper reports the results of a study to determine the most critical elements of the T_k and S_kD_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 (zeroing) of a large portion of the entries in these matrices.

In this section we describe the algorithm we used to remove values from the T_k and S_kD_k matrices and describe the impact of this sparsification strategy on retrieval quality and query run time performance.

3.1 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_kS_k matrix and the column vectors in the S_kD_k matrix can be used to describe the importance of each term (document) along a given dimension.

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_kD_k elements. However, a large portion of the T_k values can be removed without a significant change in retrieval performance.

We chose an approach that defines 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.

```
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 $NegThres$
```

Fig. 1. Sparsification Algorithm

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 the results that follow, 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.

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

The Parallel General Text Parser (PGTP) [3] 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 for all collections except OHSUMED. 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.

3.3 Impact on Retrieval Quality

The retrieval quality results for three different truncation values for the collections studied are shown in Figures 2-8. Two baselines were used to measure retrieval quality. Retrieval quality for our sparsified LSI was compared to a standard LSI system, as well as to a traditional vector space retrieval system.

Comparison to standard LSI baseline

Removal of 50% of the T_k matrix values resulted in retrieval quality that is indistinguishable from the LSI baseline for the 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 CACM have performance near or greater than the LSI baseline when 70% of the values are removed. The data for OHSUMED appears in Figure 8. Notice that sparsification at the 90% level actually improves LSI average precision by 60% at k=500.

Comparison to traditional vector space retrieval baseline

Figures 2-8 show that LSI outperforms traditional vector space retrieval for CRAN, MED and CISI even at very small values of k. However, traditional vector space is clearly better for OHSUMED, CACM, LISA and NPL at small k values. LSI continues to improve as k increases for these collections. As expected, we found no obvious relationship between the performance of the sparsified LSI system and traditional vector space retrieval. As other research studies have shown [14, 13], LSI does not always outperform traditional vector space retrieval.

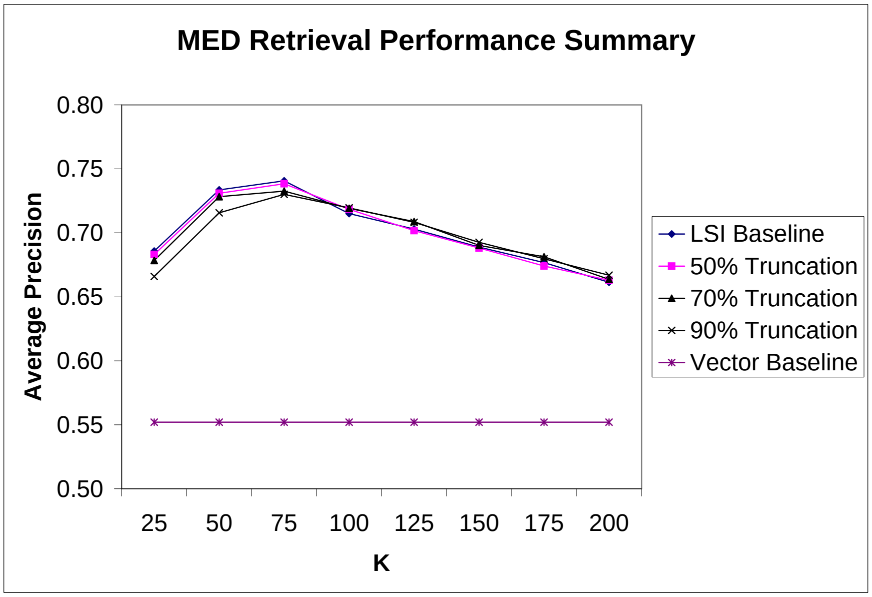

Fig. 2. Sparsification Performance Summary for MED

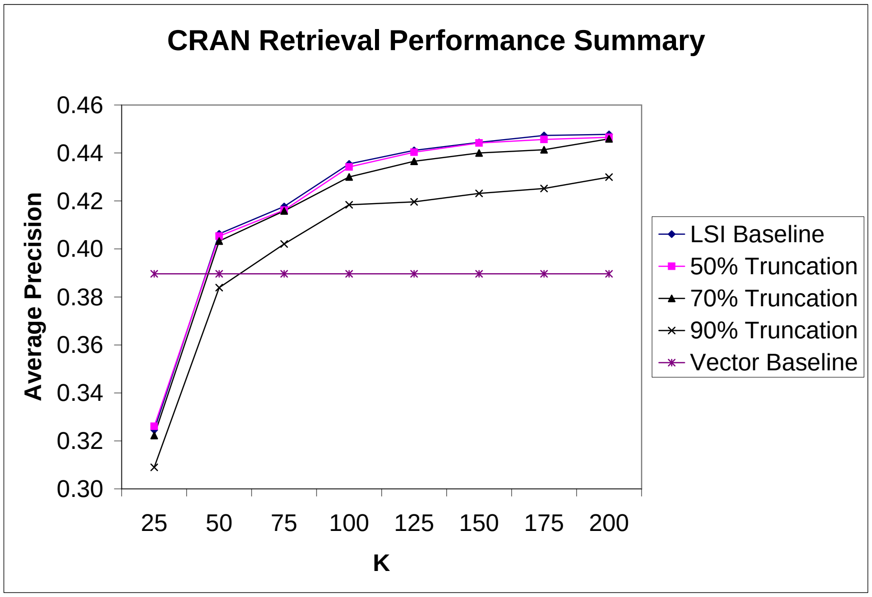

Fig. 3. Sparsification Performance Summary for CRAN

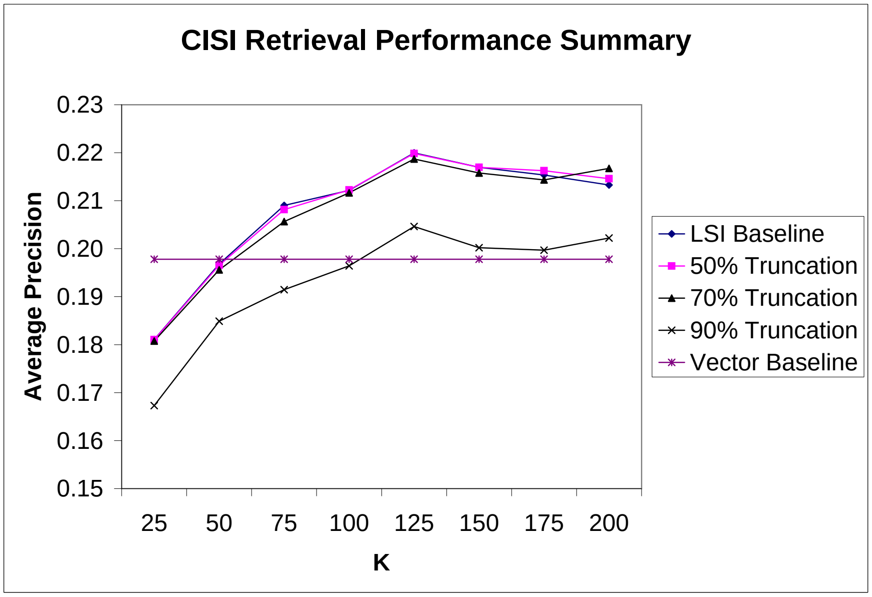

Fig. 4. Sparsification Performance Summary for CISI

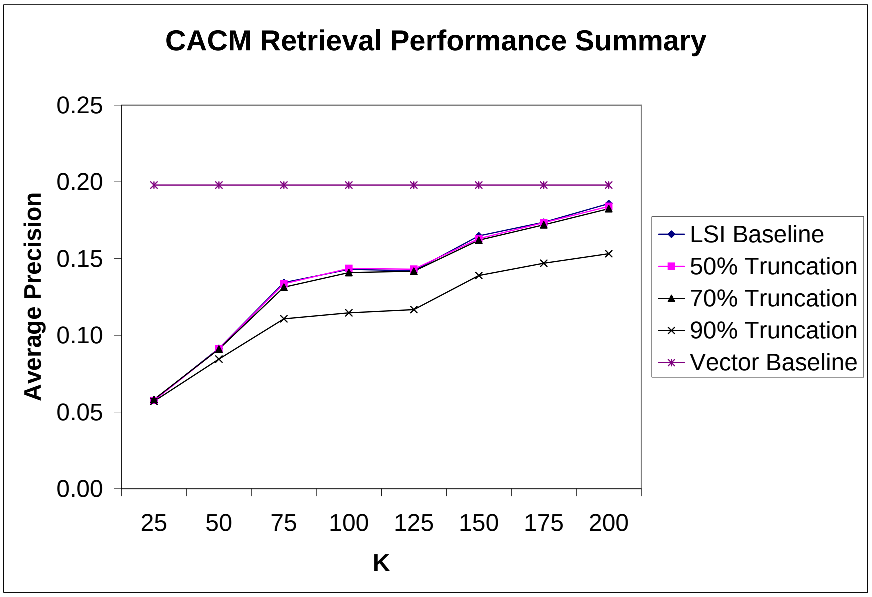

Fig. 5. Sparsification Performance Summary for CACM

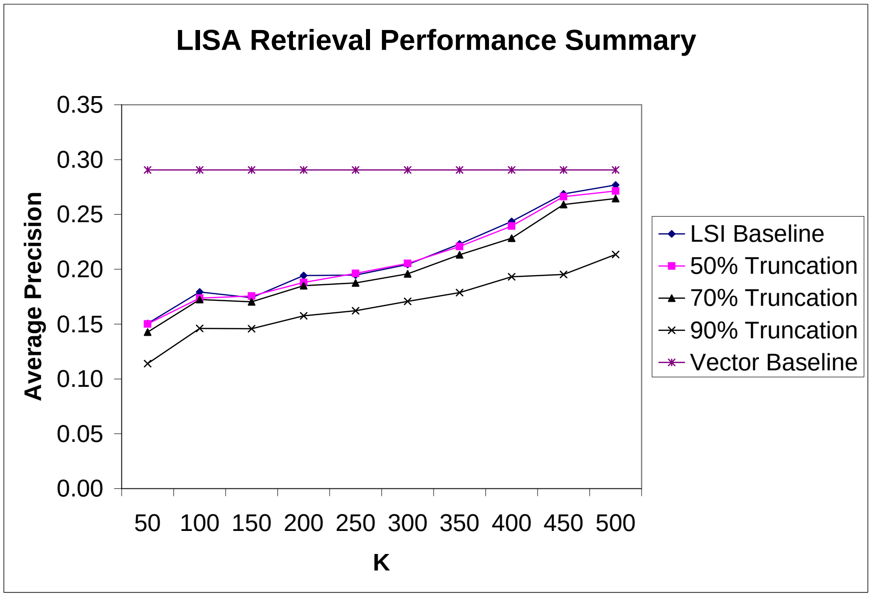

Fig. 6. Sparsification Performance Summary for LISA

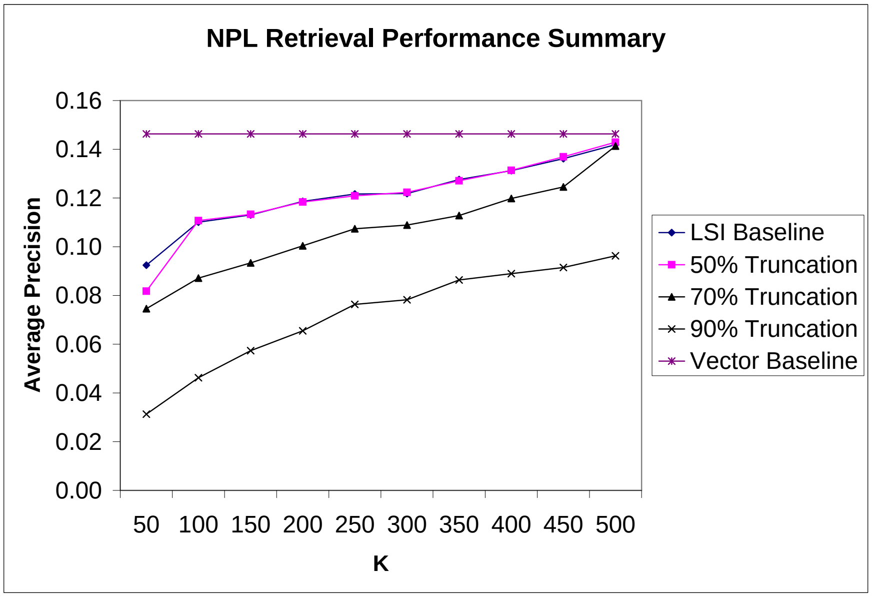

Fig. 7. Sparsification Performance Summary for NPL

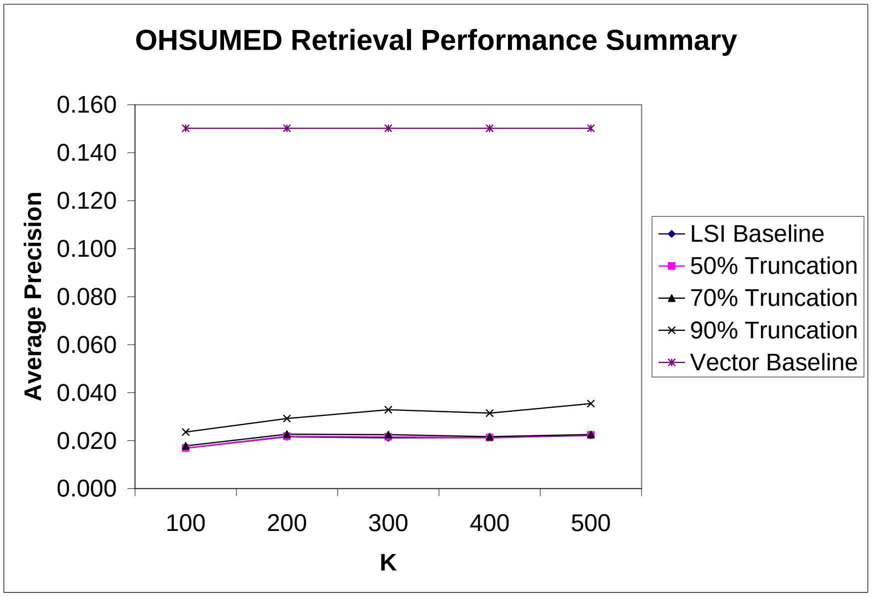

Fig. 8. Sparsification Performance Summary for OHSUMED

3.4 Impact on Runtime 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 store the new sparse matrices generated by our algorithm [7].

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 sparse matrix. Implementation of the compressed row storage format for a non-sparsified LSI system would result in an increase in the memory requirements.

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

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 $S_k D_k$ 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 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.

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

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 levels of sparsification.

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. Figures 2-8 show a slight degradation in retrieval performance (when compared with LSI) at 90% sparsification for the smaller collections; however, OHSUMED retrieval quality improves dramatically at 90% sparsification.

4 Conclusions

Our analysis has identified a large number of term and document vector values that are unimportant. This is a significant component in the development of a theoretical understanding of LSI. As researchers continue working to develop a thorough understanding of the LSI system, they can restrict their focus to the most important term and document vector entries.

Furthermore, we have shown 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.

References

1. Michael W. Berry, Zlatko Drmac, and Elizabeth R. Jessup. Matrices, vector spaces, and information retrieval. *SIAM Review*, 41(2):335–362, 1999.
2. Michael W. Berry, Susan T. Dumais, and Gavin W. O'Brien. Using linear algebra for intelligent information retrieval. *SIAM Review*, 37(4):575–595, 1995.
3. 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.
4. 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.
5. 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.

6. Chris H. Q. Ding. A similarity-based probability model for latent semantic indexing. In *Proceedings of the Twenty-second Annual International ACM/SIGIR Conference on Research and Development in Information Retrieval*, pages 59–65, 1999.
7. 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.
8. 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.
9. 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.
10. Susan T. Dumais. Using LSI for information filtering: TREC-3 experiments. In D. Harman, editor, *The Third Text REtrieval Conference (TREC-3), National Institute of Standards and Technology Special Publication 500-225*, pages 219–230, 1995.
11. George E. Forsythe, Michael A. Malcolm, and Cleve B. Moler. *Computer Methods for Mathematical Computations*. Prentice Hall, 1977.
12. 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.
13. Thomas Hofmann. Probabilistic Latent Semantic Indexing. In *Proceedings of the 22nd Annual ACM Conference on Research and Development in Information Retrieval*, pages 50–57, Berkeley, California, August 1999.
14. Parry Husbands, Horst Simon, and Chris Ding. On the use of singular value decomposition for text retrieval. In M. Berry, editor, *Proc. of SIAM Comp. Info. Retrieval Workshop*, October 2000.
15. April Kontostathis and William M. Pottenger. A framework for understanding Latent Semantic Indexing (LSI) performance. *Information Processing and Management*. To appear.
16. April Kontostathis and William M. Pottenger. Detecting patterns in the LSI term-term matrix. In *IEEE ICDM02 Workshop Proceedings, The Foundation of Data Mining and Knowledge Discovery (FDM02)*, December 2002.
17. April Kontostathis and William M. Pottenger. A framework for understanding LSI performance. In S. Dominich, editor, *Proceedings of SIGIR Workshop on Mathematical/Formal Methods in Information Retrieval*, July 2003.
18. Hinrich Schütze. Dimensions of meaning. In *Proceedings of Supercomputing*, pages 787–796, 1992.
19. Hinrich Schütze. Automatic word sense discrimination. *Computational Linguistics*, 24(1):97–124, 1998.
20. Roger E. Story. An explanation of the effectiveness of Latent Semantic Indexing by means of a Bayesian regression model. *Information Processing and Management*, 32(3):329–344, 1996.
21. Peter Wiemer-Hastings. How latent is latent semantic analysis? In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 932–937, 1999.

22. 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.
23. Hongyuan Zha and Horst Simon. A subspace-based model for Latent Semantic Indexing in information retrieval. In *Proceedings of the Thirteenth Symposium on the Interface*, pages 315–320, 1998.

Index