

Assessing the Impact of Sparsification on LSI Performance

April Kontostathis
Ursinus College
Department of Math and Computer Science
PO Box 1000, 601 Main St.
Collegeville PA 19426
akontostathis@ursinus.edu

William M. Pottenger
Lehigh University
CSE Department
19 Memorial Drive West
Bethlehem, PA 18015
billp@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 to 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 having no impact on query response time. Removal of 90% of the values results in improved query response time, but retrieval performance degrades slightly.

1. INTRODUCTION

Latent Semantic Indexing (LSI) [DDF+90] 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 [DDF+90, D93], classification [ZH01] and filtering [D94, D95]. 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 [DDF+90, D93].

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 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, which provides a natural ranking for the document set for each query.

Several outstanding issues remain with LSI. First, the SVD algorithm is computationally expensive. Many approximation algorithms have been implemented to resolve this problem [BDO+93]. Second, choosing an optimal 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 [D93]. Unfortunately, a k chosen using this technique is 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 $S_k D_k$ matrices. In the next section we describe our approach in detail and show the impact of this approach on retrieval quality and runtime performance. A similar sparsification strategy was recently presented at the Text Mining Workshop at SIAM 2003 [GZ03]. However, Gao and Zhang chose a different strategy for removing values, presented results for only one value of k , and did not provide any insight into the runtime considerations of their approach.

In future work, we hope to expand on these ideas to develop an algorithm to approximate the remaining values in the T_k and $S_k D_k$ matrices, thereby addressing the first two issues mentioned above: the computational overhead of the SVD and the optimization of the truncation value, k .

2. SPARSIFICATION OF THE LSI INPUT 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 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 and describe the impact of this sparsification strategy on retrieval quality and query run time performance.

Identifier	Description	No. of Docs	No. of Terms	No. Queries
MED	Medical Abstracts	1,033	5,831	30
CISI	Information Science Abstracts	1,460	5,143	76
CACM	Communications of the ACM Abstracts	3,204	4,863	52
CRAN	Cranfield Collection	1,398	3,931	152
LISA	Library and Information Science Abstracts	6,004	18,429	35
NPL	Larger collection of very short documents	11,429	6,988	93
OHSUMED	Clinically-oriented MEDLINE subset	348,566	170,347	106

2.1 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 [S92]. Positive values indicate a similarity between the term (document) and the concept; negative values indicate dissimilarity. Because of this association between the $T_k S_k$ and $S_k D_k$ matrix values, we decided that the best approach would define a common truncation value, based on the values in the $T_k S_k$ matrix, which would be used for both the T_k and $S_k D_k$ 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 2. We chose positive and negative threshold values that are based on the $T_k S_k$ matrix and that result in the removal of a fixed percentage of the T_k matrix. We use these values to truncate the both the T_k and $S_k D_k$ matrices.

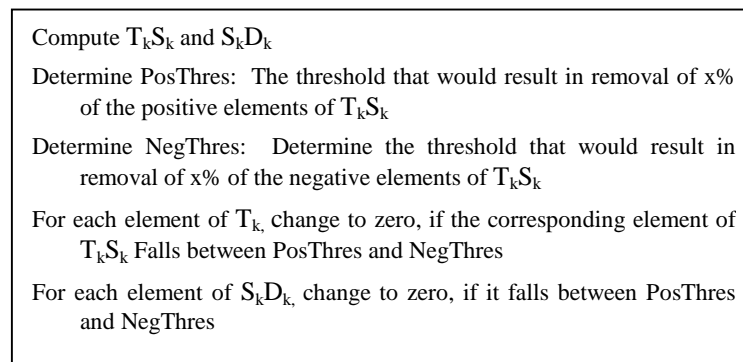


Figure 2: Sparsification Algorithm

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 [DDF+90, HK00, EW86, KHJD02]. These collections are summarized in Table 1. The Parallel General Text Parser (PGTP) [MB01] 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.

2.2 Impact on Retrieval Quality

The retrieval quality results for three different truncation values for all of the collections we studied are shown in Figure 3. 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. In the traditional vector space model, a normalized query vector is created and the cosine distance between this vector and each document vector in the term by document matrix is then computed. The cosine distance forms a natural ranking of documents for each query. The values in the query vector and the term-document matrix use the log entropy scheme, to ensure that they are consistent with the SVD produced by PGTP. The average precision values for both baselines and for the experimental technique were obtained by computing the precision of each query at recall levels .25, .50, and .75. The traditional vector space retrieval does not rely on the parameter, k , and you can see that vector average precision is shown as a constant for each collection in Figure 3. Notice the relationship between retrieval quality of LSI and that of traditional vector space across these collections. In some cases, LSI outperforms vector space retrieval (MED, CRAN, OHSUMED) even at small values of k . In other cases vector space retrieval is better (CACM, LISA, NPL), at least for the range of k values that we studied.

As you can see from Figure 3, 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, see Figure 3 and Table 4, 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. Table 2 shows the percentage of $S_k D_k$ values removed and the thresholds used for selected test cases. There are significant differences in the threshold across collections, but generally the positive and negative threshold values for a given collection/percentage combination have the similar absolute values.

Collection	K	% T_k Removed	% $S_k D_k$ Removed	Pos Threshold	Neg Threshold
MED	75	70%	23%	0.0132	-0.0135
MED	75	90%	47%	0.0286	-0.0294
CISI	125	70%	26%	0.0140	-0.0143
CISI	125	90%	53%	0.0309	-0.0318
CACM	200	70%	28%	0.0103	-0.0104
CACM	200	90%	64%	0.0295	-0.0297
CRAN	50	70%	24%	0.0160	-0.0168
CRAN	50	90%	55%	0.0408	-0.0437
LISA	500	70%	29%	0.0097	-0.0097
LISA	500	90%	66%	0.0253	-0.0250
NPL	500	70%	33%	0.0109	-0.0109
NPL	500	90%	80%	0.0350	-0.0346
OHSUMED	500	70%	9%	0.0014	-0.0014
OHSUMED	500	90%	28%	0.0045	-0.0045

Collection	k	Number of Terms	LSI (MB)	Sparsified LSI (MB)
MED	75	5831	3.3	1.5
CISI	125	5143	4.9	2.2
CACM	200	4863	7.4	3.3
CRAN	50	3931	1.5	0.7
LISA	500	18429	70.3	31.6
NPL	500	6988	26.7	12.0
OHSUMED	500	170347	649.8	292.0

Collection	k	70% Truncation		90% Truncation	
		Actual Diff Avg Prec	% Diff Avg Prec	Actual Diff Avg Prec	% Diff Avg Prec
MED	75	0.008	1%	0.011	1%
CISI	125	0.001	1%	0.000	7%
CACM	200	0.003	2%	0.033	18%
CRAN	50	0.000	2%	0.001	5%
LISA	500	0.012	4%	0.063	23%
NPL	500	0.001	0%	0.046	32%
OHSUMED	500	0.007	27%	0.016	69%

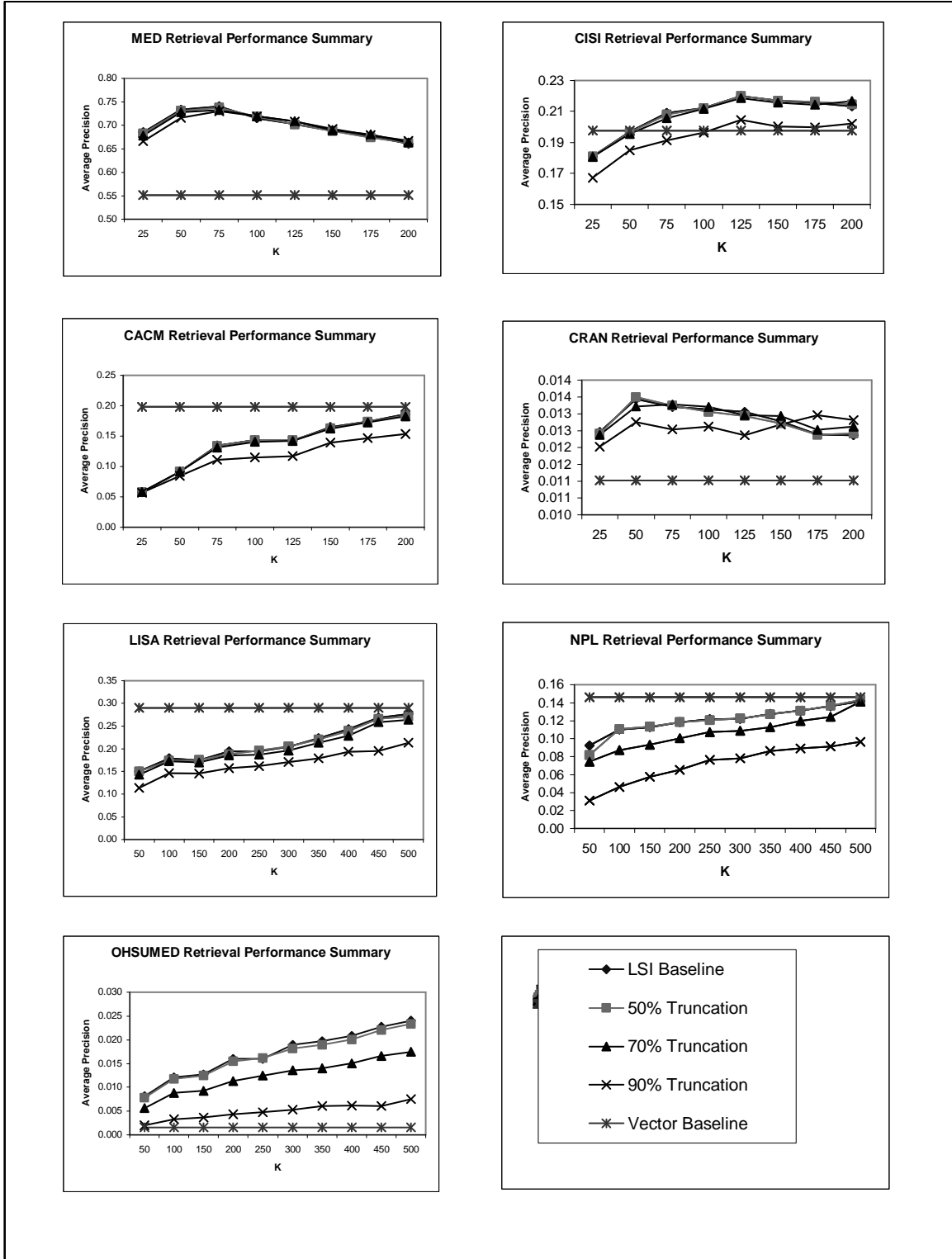


Figure 3: Retrieval Performance results for all seven collections

2.3 Impact on Query Run Time Performance

We determined that reduction of a significant percentage of the entries in the T_k and $S_k D_k$ matrices resulted in retrieval quality similar to the baseline. In this section we review the impact of sparsification on runtime performance, particularly query performance. 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 2 indicates that the S_kD_k sparsification ranges from 9% to 33%, when 70% of the T_k values are removed. A much larger S_kD_k sparsification range of 28%-88% is achieved at a 90% reduction in the T_k matrix, but retrieval quality is degraded at the 90% level (see Table 4).

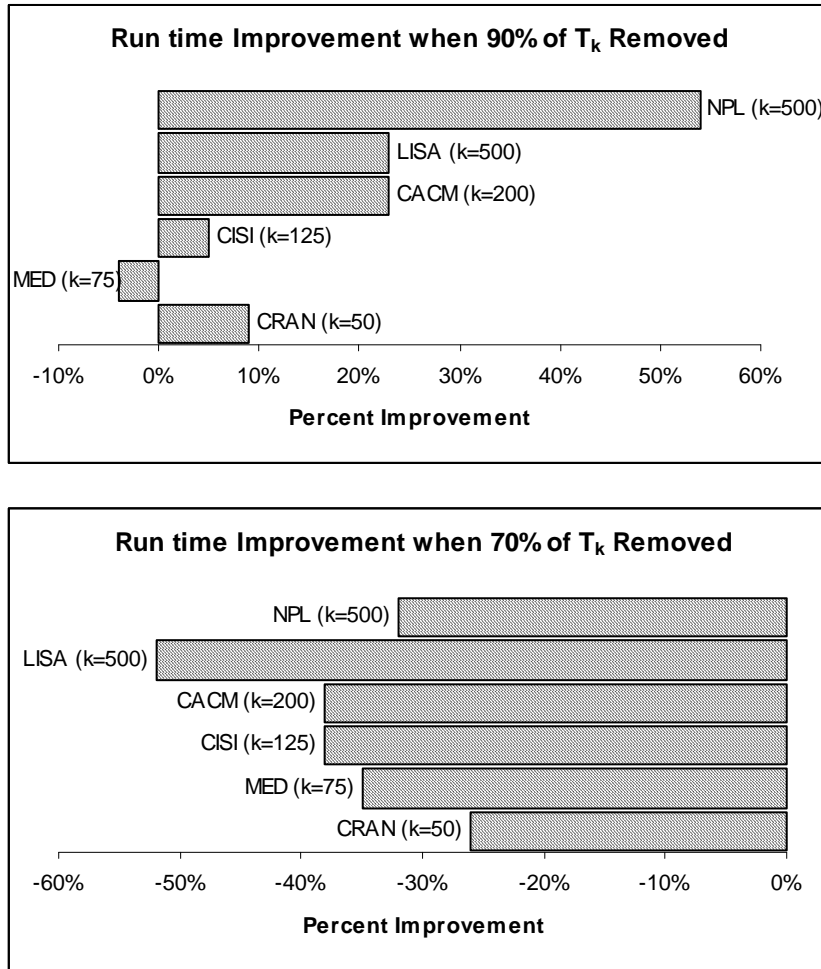


Figure 4: Impact of Sparsification on query run time performance for selected test cases

In order to determine the impact of sparsification on query run time performance we implemented a sparse matrix version of the LSI query processing. A very simple sparse matrix format was used, implementing parallel vectors for the data and index values for the vectors that comprise the T_k and S_kD_k matrices. The Standard Template Library (STL) implementation of vectors was used, and no attempt was made to optimize the performance of either the full vector or the sparse vector implementation of the query processing. Preliminary testing revealed no differences in performance (comparing the baseline LSI to the sparsified LSI) when either full vectors were used for both the T_k and S_kD_k matrices or when the sparse vector implementation was used for the T_k vectors only. Only when we implemented the sparse matrix solution for the S_kD_k matrix did we discover a difference in run time performance.

The runtime data was compiled using an SGI Origin 3800 32-way SMP, running IRIX 6.5. Main memory size is 32 Gbytes, with an instruction cache size of 32 Kbytes, and a data cache of 32 Kbytes. The secondary unified instruction/data cache size is 8 Mbytes. The runs utilized a queue that dedicated a processor to the program for the entire duration of the run. 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 run twice and the results in Figure 4 represent the average of the two runs for selected records.

Sparsification of 90% of the T_k elements results in an improvement in query runtime performance for all collections except MED, while the runtime performance of the sparsified code is degraded when a 70% sparsification threshold is applied. The data collected was consistent across all values of k studied.

3. CONCLUSIONS AND FUTURE WORK

We conclude from this data that query run time improvements in LSI can be achieved using our sparsification strategy only if degradation in retrieval quality can be tolerated, see Table 4 for the retrieval quality summary for all collections at selected values of k . We note, however, that query run time performance remains constant if the full vector version of the $S_k D_k$ matrix is implemented. In this case, values between the positive and negative thresholds are set to zero and the sparse vector format is implemented for the T_k vectors, resulting in a significant memory savings for the entire LSI system, as quantified in Table 3.

In future work, we will to expand on these ideas to develop an algorithm to approximate the remaining values in the T_k and $S_k D_k$ matrices. LSI is computationally expensive [BDO+93]; and an efficient algorithm that approximates the values in these matrices would be a valuable tool for information retrieval.

4. ACKNOWLEDGMENTS

This work was supported in part by National Science Foundation Grant Number EIA-0087977. The authors also would like to express their gratitude to Dr. Brian Davison for his comments on the draft. The authors also gratefully acknowledge the assistance of their colleagues in the Computer Science and Engineering Department at Lehigh University in completing this work. 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.

5. REFERENCES

- [BDJ99] Berry, M., Z. Drmac and E.R. Jessup. 1999. Matrices, Vector Spaces, and Information Retrieval. *SIAM Review*41:2. 1999. pp. 335-362
- [BDO+93] Berry, M., T. Do, G. O'Brien, V. Krishna, and S. Varadhan. 1993. SVDPACKC (Version 1.0) User's Guide. April 1993.
- [BDO95] Berry, M. W., Dumais, S. T., and O'Brien, G. W. 1995. "Using linear algebra for intelligent information retrieval." *SIAM Review*, 37(4), 1995, 573-595.
- [DDF+90] Deerwester, S., S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman. 1990. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6): 391-407.
- [D93] Dumais, S. T. 1993. LSI meets TREC: A status report. *The First Text REtrieval Conference (TREC1)*, D. Harman (Ed.), *National Institute of Standards and Technology Special Publication 500-207*, pp. 137-152.
- [D94] Dumais, S. T. 1994. Latent Semantic Indexing (LSI) and TREC-2. In: D. Harman (Ed.), *The Second Text REtrieval Conference (TREC2)*, *National Institute of Standards and Technology Special Publication 500-215*, pp. 105-116
- [D95] Dumais, S. T. 1995. Using LSI for information filtering: TREC-3 experiments. In: D. Harman (Ed.), *The Third Text REtrieval Conference (TREC3)* *National Institute of Standards and Technology Special Publication*, in press 1995.
- [EW86] El-Hamdouchi, A. and P. Willet. 1986. Hierarchic document clustering using ward's method, *In proceedings of the Ninth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*.
- [GZ03] Gao, J. and J. Zhang. 2003. Sparsification Strategies in Latent Semantic Indexing. *Proceedings of the 2003 Text Mining Workshop*. M. W. Berry and W. M. Pottenger, Eds. pp. 93-103.
- [HK00] Han, E.H., and G. Karypis: 2002. Fast Supervised Dimensionality Reduction Algorithm with Applications to Document Categorization & Retrieval. *Proceedings of CIKM 2000*: pp. 12-19.
- [KHJD02] Klink, S, A. Hust, M. Junker, and A. Dengel. 2002. Improving Document Retrieval by Automatic Query Expansion Using Collaborative Learning of Term-Based Concepts. In *Document Analysis Systems V*. D. Lopresti, J. Hu, and R. Kashi (Eds.).
- [MB01] Martin, D. I. and M. W. Berry. 2001. Parallel General Text Parser. Copyright 2001.
- [S92] Schütze, H. 1992. Dimensions of Meaning. *Proceedings of Supercomputing '92*.
- [W99] Wiemer-Hastings, P. 1999. How Latent is Latent Semantic Analysis? *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, Stockholm, Sweden, Aug 1999, pp. 932-937. San Francisco: Morgan Kaufmann.
- [ZH01] Zelikovitz, S. and H. Hirsh. 2001. Using LSI for Text Classification in the Presence of Background Text. *Proceedings of CIKM-01, 10th ACM International Conference on Information and Knowledge Management*.