Divide and Conquer Strategies for Effective Information Retrieval*

Jie Chen† Yousef Saad†

Abstract

The standard application of Latent Semantic Indexing (LSI), a well-known technique for information retrieval, requires the computation of a partial Singular Value Decomposition (SVD) of the term-document matrix. This computation becomes infeasible for large document collections, since it is very demanding both in terms of arithmetic operations and in memory requirements. This paper discusses two divide and conquer strategies, with the goal of alleviating these difficulties. Both strategies divide the document collection in subsets, perform relevance analysis on each subset, and conquer the analysis results to form the query response. Since each sub-problem resulting from the recursive division process has a smaller size, the processing of large scale document collections requires much fewer resources. In addition, the computation is highly parallel and can be easily adapted to a parallel computing environment. To reduce the computational cost, we perform the analysis on the subsets by using the Lanczos vectors instead of singular vectors as in the standard LSI method. This technique is far more efficient than the computation of the truncated SVD, while its accuracy is comparable. Experimental results confirm that the proposed divide and conquer strategies are effective for information retrieval problems.

1 Introduction

Techniques of information retrieval extract relevant documents in a collection, in response to a user query. As is well-known [2] information retrieval techniques based on exact literal matching, i.e., on direct comparisons between the columns of the term-document matrix and the query, may be inaccurate due to common problems of word usage such as *synonymy* and *polysemy*. *Latent Semantic Indexing* (LSI) [12] is a well-known method which was developed to deal with these difficulties. LSI projects the original term-document matrix into a reduced rank subspace by resorting to the Singular Value Decomposition (SVD). The comparison of the query with the documents is then performed in this subspace and produces in this way a more meaningful result.

To be specific, let a collection of m terms and n documents be represented in an $m \times n$ term-document matrix

$$X = [x_{ij}]$$

where x_{ij} is the *weight* of term i in document j. The weights x_{ij} depend on how often the term i appears in document j but also on other scalings used, see, e.g., [2, 11] for details. A term-document matrix is generally very sparse because a given term typically occurs only in a small subset of documents. A query is represented as a *pseudo-document* in a similar form, $q = [q_i]$, where q_i represents the weight of term i in the query.

The *Vector Space Model* (VSM) computes the similarity between the query q and a document vector x_j, which is the j-th column of X, simply as the cosine of the angle between the two vectors:

$$\cos(q, x_j) = \frac{q^T x_j}{\|q\|_2 \, \|x_j\|_2}.$$

With this approach, we can rank all documents in the collection with respect to their relevance to q by simply computing the n-dimensional vector

$$s = q^T X \tag{1.1}$$

and scaling the results with the norms of the corresponding columns of X.

The model just described is too simple and it is ineffective in practice as it relies on exact litteral matches between entries in q and those in X. LSI addresses this problem by projecting the data matrix X into a small dimensional subspace with the help of the SVD. To be specific, let X have the SVD $X = U\Sigma V^T$, with its truncated rank-k version:

$$Y_k = U_k \Sigma_k V_k^T, \tag{1.2}$$

where U_k (resp. V_k) consists of the first k columns of U (resp. V), and Σ_k is the k-th principal submatrix of Σ. The matrix Y_k is the best rank-k approximation of X in the 2-norm or Frobenius norm sense [13, 17]. Up to scalings, the relevance vector produced by LSI is then given by

$$s_k = q^T Y_k, \tag{1.3}$$

*This work was supported by NSF grants DMS 0510131 and DMS 0528492 and by the Minnesota Supercomputing Institute.

†Department of Computer Science and Engineering, University of Minnesota at Twin Cities, MN 55455. `{jchen, saad}@cs.umn.edu`.

which replaces the expression (1.1) of the VSM.

Current implementations of LSI mainly rely on matrix decompositions (see e.g., [4, 23]), predominantly the truncated SVD [2, 3]. A notorious difficulty with this approach is that it is computationally expensive for large X. In addition, frequent changes in the data require updates to the SVD, and this is not an easy task. Much research has been devoted to the problem of updating the (truncated) SVD [29, 31, 7, 27]. A drawback of these approaches is that the resulting (truncated) SVD loses accuracy after frequent updates.

To bypass the truncated SVD computation, Kokiopoulou and Saad [22] introduced polynomial filtering techniques, and Erhel *et al.* [14] and Saad [26] proposed algorithms for building good polynomials to use in such techniques. These methods all efficiently compute a sequence of vectors that progressively approximate the vector s_k defined in (1.3), without resorting to the expensive SVD.

In this paper we propose two divide and conquer techniques with a primary goal of reducing the cost of LSI. The divide and conquer strategy is particularly attractive for very large problems when performing the SVD is difficult. The proposed methods recursively divide the data set using spectral bisection techniques, and separately perform relevance analysis on each subset. Then, partial analysis results are conquered to form the final answer. The two proposed strategies differ in how the data set is partitioned, as well as in the subsequent conquering process.

One advantage of dividing the term-document matrix X into smaller subsets is that the analysis of each subset becomes feasible, say on a single machine, even when X is large. In a parallel environment, all the subsets can be analyzed in parallel and the partial results can be combined. Furthermore, the computation and analysis on smaller subsets will be much cheaper than that on X itself.

Though the primary motivation for the proposed strategies lies in their computational efficiency, they are nevertheless designed by exploiting the underlying semantics of the texts. Our divide steps are based on a spectral bisection method [6], which has been proven to be effective for clustering tasks in text mining applications. Since each resulting subset has more homogeneous contents than the original data set, analysis on these subsets tends to extract more semantic related information and structures than the LSI method on the whole set. This idea of breaking an inhomogeneous data set into homogeneous subsets was also investigated in [16], where the authors performed a k-means clustering on the documents before running LSI on each cluster. This paper uses a much more efficient partitioning technique, and shows that the idea applies to not only documents, but also terms.

Another feature included in the proposed strategies is that we use Lanczos vectors to perform relevance analysis on the subsets resulting from the divide processes. This is an important step to significantly reduce the computational cost. As is well known in numerical linear algebra, the Lanczos procedure [24] is often preferred for computing the largest k singular vectors of a large sparse matrix [5]. Nevertheless, it takes a large number of iterations to obtain accurate singular vectors when k is large (as in the case for LSI in practice). Instead, we run only k Lanczos iterations and do not compute any singular vectors. We use the k Lanczos vectors to perform the relevance analysis on each subset. It is shown in [10] that using these k Lanczos vectors is an effective replacement of the k singular vectors for dimensionality reduction.

The rest of the paper is organized as follows. Section 2 presents the two divide and conquer strategies, and Section 3 discusses the efficient analysis on the subsets. These two sections combined constitute the main algorithmic contributions of the paper. Then in Section 4 a few experiments are shown, and concluding remarks are given in Section 5.

2 Divide and conquer strategies

A paradigm known for its effectiveness when solving very large scale scientific problems is that of multilevel approaches. The term "multilevel" can have different meanings depending on the application and general methodology being used. In the context of linear systems of equations, powerful strategies, specifically multigrid or algebraic multigrid methods [9, 8, 18] have been devised to solve linear systems by essentially resorting to divide and conquer strategies that exploit the relationship between the mesh and the eigenfunctions of the operator.

In the context of information retrieval, divide and conquer strategies will consist of splitting the original data into smaller subsets on which the analysis of similarities between the query and the documents is performed. We consider two ways to perform divide and conquer in this section. The first one, *column partitioning*, is generally useful for data matrices X with many more rows than columns, while the second one, *row partitioning*, is for matrices that have many more columns than rows. Note that this is somewhat counter-intuitive. Indeed, one is inclined to partitioning the column set if the matrix is wide and short, and the row set in the opposite situation. However, a little cost analysis along with experimentation shows that the opposite is computationally more appealing.

In a nutshell, we recursively bisect (either vertically or horizontally) the data matrix using an efficient spectral bisection technique, and accordingly design methods to compute the query relevance. From the point of view of data clustering, the partitioning exposes more latent semantic structures than the original data matrix, since the data (documents or terms) are clustered according to the common concepts they represent. On the other hand, the low-rank-plus-shift theorem (see Section 2.2.2) indicates that performing analysis on individual subsets has a similar effect to that on the whole set, since latent structures are preserved. With these guarantees, we focus on how to effectively perform the analysis on subsets and produce accurate relevance scores.

2.1 Divide and conquer on the document set (column partitioning). It may be most natural to think of grouping documents in subsets by invoking similarities between the documents. Given a set of n documents represented in matrix form as $X = [x_1, x_2, \ldots, x_n] \in \mathbb{R}^{m\times n}$, the top-down approach to clustering the document set is to partition the set (of columns) recursively in two subsets until a desirable number of clusters is reached. This partitioning can be done in a number of ways. We select *spectral bisection*, a simple-to-implement technique whose effectiveness has been well documented in the literature, see, e.g., [6, 28, 21, 15].

The main ingredient used by spectral techniques in order to divide a set in two is the property that the largest left singular vector[1] u of $\bar{X} = X - ce^T$ yields a hyperplane which separates the set X in two good clusters, namely

$$\text{(2.4a)} \qquad \begin{cases} X_+ = \{x_i \mid u^T(x_i - c) \geq 0\}, \\ X_- = \{x_i \mid u^T(x_i - c) < 0\}. \end{cases}$$

Here c is the centroid of the data set and e is the column vector of all ones. Equivalently, this is to split the set into subsets

$$\text{(2.4b)} \qquad \begin{cases} X_+ = \{x_i \mid v_i \geq 0\}, \\ X_- = \{x_i \mid v_i < 0\}, \end{cases}$$

where v is the largest right singular vector of $\bar{X}$. If it is preferred that the sizes of the clusters are balanced, an alternative is to replace the above criterion by

$$\text{(2.5)} \qquad \begin{cases} X_+ = \{x_i \mid v_i \geq \mathrm{m}(v)\}, \\ X_- = \{x_i \mid v_i < \mathrm{m}(v)\}, \end{cases}$$

[1]By abuse of language we will use the term *largest singular vector* to mean the singular vector associated with the largest singular value.

where $\mathrm{m}(v)$ represents the median of the entries of vector v.

Computationally, the largest left/right singular vectors can be inexpensively computed via the Lanczos algorithm by exploiting the sparsity of X [6]. Note that from the point of view of graph partitioning, the technique just described partitions the set of columns without resorting to the graph Laplacian. Therefore, we are implicitly partitioning the *hypergraph* which is canonically associated with the matrix X when the hyperedges are defined from the columns of X.

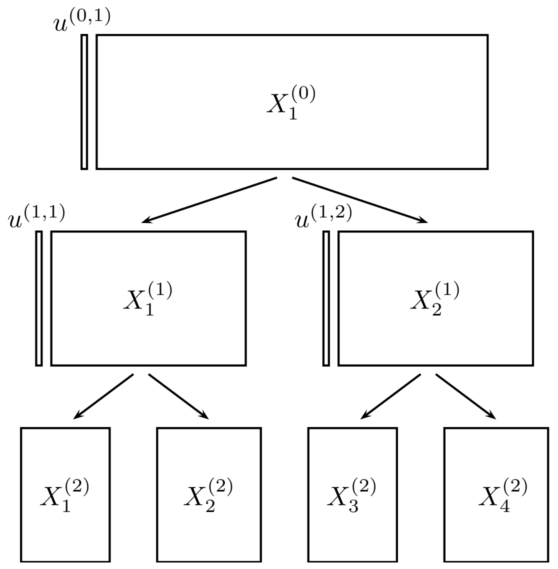


Figure 1: A 2-level subdivision of a term-document matrix X. The vectors $u^{(i,j)}$ shown for levels 0 and 1 are perpendicular to the separating hyperplanes associated with the nodes. Note that this figure is not drawn to scale—column partitioning is better applied to matrices that are tall and thin.

The bisection tool just described can be employed to recursively divide the data set; see Figure 1 for an illustration. At level zero (initial level) we have only one set $X_1^{(0)} \equiv X$. This set is partitioned into two subsets $X_1^{(1)}$ and $X_2^{(1)}$, which are further partitioned into $X_1^{(2)}$, $X_2^{(2)}$, $X_3^{(2)}$ and $X_4^{(2)}$. A binary tree structure results from this recursive partitioning procedure, which selects the largest leaf node to partition each time until a desired number of leaf nodes are created. Note that this tree need not be a complete tree, i.e., all the leaves might not be at the same level.

At each node in the tree, a vector of length m (denoted by $u^{(i,j)}$ in the figure) is needed when a further subdivision of the set $X_j^{(i)}$ is required. Indeed, this node can only be a non-leaf node in the final tree. The vector $u^{(i,j)}$ is the largest left singular vector of $X_j^{(i)}$ after centering. The separating hyperplane associated with this node has a normal in direction $u^{(i,j)}$ and passes through the centroid $c^{(i,j)}$ of $X_j^{(i)}$. The vector $u^{(i,j)}$ always points towards the left child of $X_j^{(i)}$. For consistency, we label the two children $X_{2j-1}^{(i+1)}$ and $X_{2j}^{(i+1)}$. See Figure 2 for an illustration.

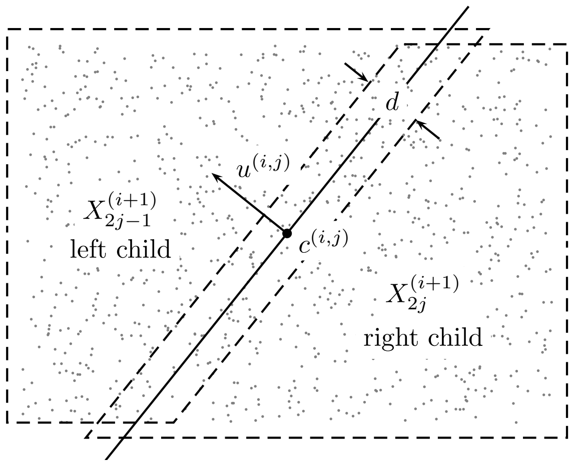


Figure 2: Illustration of partitioning a node $X_j^{(i)}$ into two children $X_{2j-1}^{(i+1)}$ and $X_{2j}^{(i+1)}$. The vector $u^{(i,j)}$ always points towards the left child. The two dashed frames indicate the actual configuration of the two subsets, which share a margin of width d around the hyperplane.

In practice we create a margin around the separating hyperplane, and both children share this margin. This small overlapping of the subsets helps improve accuracy. The analysis on both sides of the hyperplane may result in discrepancies for the similarity between a document inside the margin and a specific query. Overlapping the two children allows flexibility when deciding the relevance scores of the documents inside the margin.

2.1.1 Query response. Given a query q, the standard LSI works by performing analysis on the whole data set X. Specifically, the k largest singular vectors of X are extracted and they form a subspace with regard to which the similarities between q and columns of X are computed. In our divide and conquer strategy, X is partitioned into several subsets (leaf nodes), hence analysis can be performed on each leaf node and relevance scores are produced. In Section 3 we will see an efficient way to perform this analysis. In this section, let us temporally disregard the analysis and simply assume that relevance scores are available.

If a document appears in only one leaf node, the relevance score for this document is just the one resulting from the analysis on this node. If it appears in multiple leaf nodes, we select the maximum of all scores from these nodes to be the relevance score for this document. A document appears in multiple leaf nodes because it is residing within the margin of some dividing hyperplane. The recursive partitioning procedure essentially performs clustering on the whole document collection, hence the document being considered is on the border of several clusters. The question "how relevant this document is to the query" may yield different answers if the analysis is separately performed on each clusters. The maximum score from the analysis captures the most probable relevance of the document to the given query.

This divide and conquer strategy based on column partitioning is summarized in Algorithm 1.

Algorithm 1 D&C-LSI: Column Partitioning

```
1: Recursively bisect the columns of $X$. A binary tree structure is formed as in Figure 1.
2: For a given query $q$, compute relevance scores between $q$ and those in the leaf nodes.
3: The relevance score of a document is the maximum of all the scores computed in the previous step for this document.
```

2.2 Divide and conquer on the term set (row partitioning). It is also possible to partition terms instead of documents and this can be a better approach than that of the preceding subsection when $m \le n$. The partitioning approach is identical to that of the previous subsection with the simple exception that it is applied to the transpose of X.

The matrix X is now partitioned row-wise into p subsets and the query q is split accordingly, i.e.,

$$X = \begin{bmatrix} X_1 \\ \vdots \\ X_p \end{bmatrix} \quad \text{and} \quad q = \begin{bmatrix} q_1 \\ \vdots \\ q_p \end{bmatrix}. \tag{2.6}$$

See Figure 3 for an illustration. Here each X_i represents a matrix of size $m_i \times n$ containing m_i rows of X, obtained by a recursive spectral bisection technique on the rows rather than the columns. For example, $X_1, \ldots, X_4$ in Figure 3 can be obtained the same way as shown in Figure 1, with the illustration rotated 90° counter clockwise. For simplicity, here we use 1, 2, ..., p to index all the leaf nodes instead of using the (i, j) tuple to index the subsets in the hierarchy as in the previous subsection.

2.2.1 Query response. The question now is how to compute the similarities between a query and the

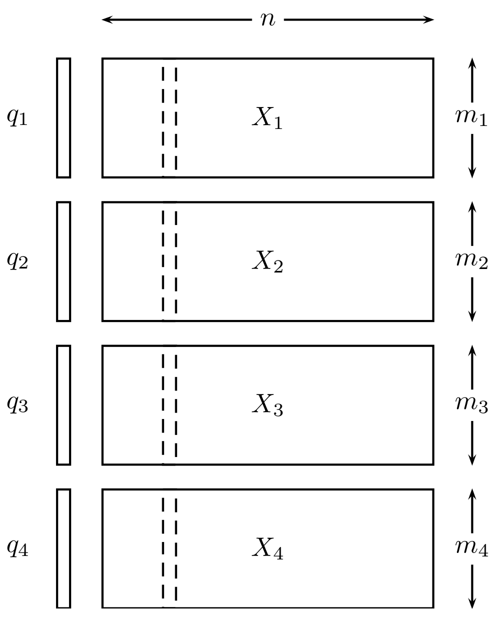


Figure 3: A term-wise subdivision of a term-document matrix X into 4 subsets. The query q is subdivided accordingly. Note that this figure is not drawn to scale—row partitioning is better applied to matrices that are wide and short.

columns of the original matrix X. If we had just computed the inner products of the partial columns of X with the corresponding parts of q and then added the results, the result would be no different from that of the standard VSM. We need instead to perform an analysis similar to LSI on each subdivision of X and combine the partial results. In this section we illustrate how this is done using the truncated SVD approach. This approach can be easily modified into a more efficient one based on the Lanczos algorithm as described in Section 3.

Let the local SVD of each X_i be $X_i = U_i \Sigma_i V_i^T$ with its truncated rank-k version

$$Y_{i,k} = U_{i,k} \Sigma_{i,k} V_{i,k}^T. \tag{2.7}$$

Then, the similarity scores for all the documents are

$$s'_k = \sum_{i=1}^{p} q_i^T Y_{i,k} \tag{2.8}$$

followed by a scaling with the norms of the corresponding columns of

$$\hat{X} = \begin{bmatrix} Y_{1,k} \\ \vdots \\ Y_{p,k} \end{bmatrix}.$$

Note that formula (2.8) is very similar to (1.3), except that both q and X are partitioned.

The sparsity of the query vector can be exploited to reduce the computational cost. It is likely that some parts of q are completely zero, hence it is not necessary to compute $q_i^T Y_{i,k}$ for some i's. The cost will be greatly reduced if only one or two subdivisions of q have non-zero elements, which is common in practice.

This divide and conquer strategy based on row partitioning is summarized in Algorithm 2. Note that the analysis in lines 3 and 4 will be replaced later by an efficient alternative described in Section 3.

Algorithm 2 D&C-LSI: Row Partitioning

```
1: Recursively bisect the rows of $X$, resulting in submatrices $X_1, X_2, \ldots, X_p$.
2: For a given query $q$, subdivide the rows of $q$ accordingly.
3: Perform analysis on each subdivision and sum up the results as in (2.8).
4: Scale each entry of $s'_k$ by the norm of the corresponding column of $\hat{X}$.
```

2.2.2 Rationale of the use of $\hat{X}$. Comparing Equation (2.8) with (1.3), this divide and conquer strategy essentially performs an analysis with the matrix $\hat{X}$ replacing X in the standard LSI. The validity of this approach comes from the fact that the best rank-k approximation of X and that of $\hat{X}$ are the same under certain conditions. All this means is that we do not lose latent information by performing analysis on each individual subdivision of X instead of on X itself.

The proof that the two approximations just mentioned are equal is based on the following theorem:

THEOREM 2.1. *Let* $\text{best}_k(\cdot)$ *denote the best rank-k approximation of a matrix. Consider*

$$X = \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} \quad \text{and} \quad \hat{X} = \begin{bmatrix} \text{best}_k(X_1) \\ \text{best}_k(X_2) \end{bmatrix},$$

where $X_1 \in \mathbb{R}^{m_1 \times n}$ *and* $X_2 \in \mathbb{R}^{m_2 \times n}$ *with* $m_1 + m_2 \le n$. *Assume that there exists a symmetric positive semi-definite matrix* Y *of rank* k *and a positive number* σ *such that* $XX^T = Y + \sigma I$, *then*

$$\text{best}_k(X) = \text{best}_k(\hat{X}).$$

This theorem is essentially the row-wise version of Theorem 4.2 in [32]. (See also the remark thereafter.) Hence the proof is omitted here. Note that the theorem is restricted to situations when $m \le n$, since otherwise XX^T is singular.

Theorem 2.1 indicates that if XX^T has a *low-rank-plus-shift* structure, then the best rank-k approximation of X is the same as that of $\hat{X}$. Hence by induction, when X is subdivided in p partitions as in (2.6), then $\text{best}_k(X) = \text{best}_k(\hat{X})$ still holds, assuming that X, as well as all the submatrices to be partitioned in the subdivision process, has the low-rank-plus-shift structure when being multiplied by its transpose.

As pointed out in [32] after a perturbation analysis, it is not unrealistic to assume the low-rank-plus-shift structure for real-life data. Hence it is reasonable to use $\hat{X}$ in place of X in a divide and conquer analysis. As was mentioned above, the theorem implicitly assumes that $m \leq n$, that is, the term-document matrix X is wide and short. Though this is not true for many small-scale existing experimental data sets, one should expect that in practice it is the dominant situation, since vocabulary is limited while text corpora are growing in size. We use two such data sets in Section 4 for experiments.

2.3 More divide and conquer strategies. The two partitioning strategies previously described can be combined to devise other divide and conquer strategies. When X itself is considered a hypergraph, the subdivision of X amounts to partitioning the edges or vertices of the graph. A number of articles applying hypergraph partitioning techniques, e.g., [19, 1], have exploited the sparsity of X and considered reordering the rows and columns of X into block form or near block form. Then the data matrix X is naturally partitioned into blocks according to the sparsity patterns, and analysis can be performed on the diagonal blocks. An illustration is given in Figure 4. In this paper we will not consider this reordering approach further and leave it for future investigations.

3 Relevance analysis using Lanczos vectors

As mentioned earlier, the LSI method, which uses singular vectors of the term-document matrix for relevance analysis, is expensive due to the computation of the truncated SVD. Instead, we propose using a technique, called *Lanczos approximation*, to perform relevance analysis in our divide and conquer strategies. This technique essentially computes the query relevance in the subspace spanned by the Lanczos vectors instead of that spanned by the singular vectors. The effectiveness of this technique when applied to the whole data set X has been demonstrated in a report [10]. In this paper we apply this technique to each subdivision of X. We first briefly review the symmetric Lanczos algorithm, which is later used to develop the relevance analysis.

3.1 The symmetric Lanczos algorithm. Given a symmetric matrix $A \in \mathbb{R}^{n\times n}$ and an initial unit vector

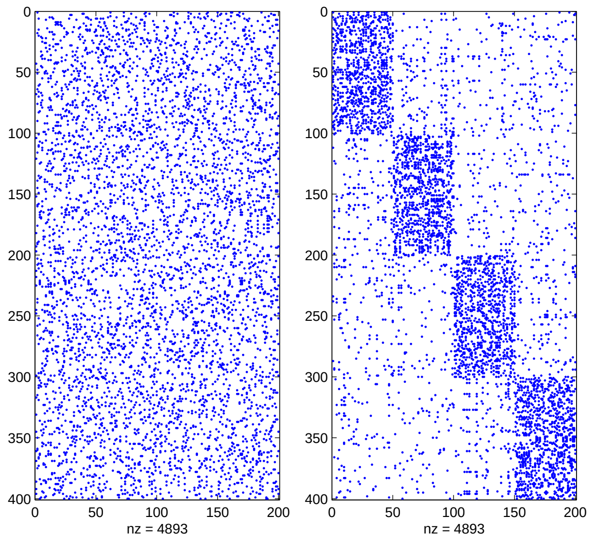


Figure 4: A small rectangular sparse matrix before and after reordering in block diagonal form.

q_1, the Lanczos algorithm builds an orthonormal basis of the Krylov subspace

$$\mathcal{K}_k(A, q_1) = \text{span}\{q_1, Aq_1, A^2q_1, \ldots, A^{k-1}q_1\}.$$

The vectors q_i, $i = 1, \ldots, k$ computed by the algorithm satisfy the 3-term recurrence

$$\beta_{i+1}q_{i+1} = Aq_i - \alpha_i q_i - \beta_i q_{i-1}$$

with $\beta_1 q_0 \equiv 0$. The coefficients α_i and β_{i+1} are computed so as to ensure that $\langle q_{i+1}, q_i\rangle = 0$ and $\|q_{i+1}\|_2 = 1$. In exact arithmetic, it turns out that q_{i+1} is orthogonal to $q_1, \ldots, q_i$ so the vectors q_i, $i = 1, \ldots, k$ form an orthonormal basis of the Krylov subspace $\mathcal{K}_k(A, q_1)$. In practice some form of reorthogonalization is needed, as orthogonality is lost fairly soon in the process.

If $Q_k = [q_1, \ldots, q_k] \in \mathbb{R}^{n\times k}$ then an important equality resulting from the algorithm is

$$Q_k^T A Q_k = T_k = \begin{bmatrix} \alpha_1 & \beta_2 & & & \\ \beta_2 & \alpha_2 & \beta_3 & & \\ & \ddots & \ddots & \ddots & \\ & & \beta_{k-1} & \alpha_{k-1} & \beta_k \\ & & & \beta_k & \alpha_k \end{bmatrix}.$$

An eigenvalue θ of T_k is called a *Ritz value*, and if y is an associated eigenvector, $Q_k y$ is called the associated *Ritz vector*. As k increases more and more Ritz values and vectors will converge towards eigenvalues and vectors of A [17, 25].

3.2 Lanczos approximation. A good alternative to the vector s_k defined in (1.3) can be efficiently obtained by exploiting the Lanczos procedure. Let $A = X^T X$ be applied to the Lanczos algorithm which yields equality

$$Q_k^T X^T X Q_k = T_k. \tag{3.9}$$

By defining

$$w_i := q^T X Q_i Q_i^T, \quad i = 1, 2, \ldots \tag{3.10}$$

i.e., letting w_i be the projection of $s = q^T X$ onto the subspace range(Q_i), it can be proved that the difference between w_i and s in the right singular direction v_j of X decays at least with the same order as $T_{i-j}^{-1}(\gamma_j)$, where $T_{i-j}(\cdot)$ is the Chebyshev polynomial of the first kind of degree $i - j$, and γ_j is a constant independent of i and larger than 1. This means that as i increases, w_i is progressively closer to s in major singular directions of X. Hence when $i = k$, w_i is considered a good approximation to s_k.

The vector w_k can be most efficiently computed via matrix-vector multiplications $((q^T X)Q_k)Q_k^T$. The computation of w_k in this way is much more inexpensive than the computation of $s_k = q^T X_k$. In fact, computing s_k requires computing the truncated SVD of X. A typical way of computing X_k is to go through the Lanczos process as in (3.9) with $k' > k$ iterations, and then compute the eigen-elements of $T_{k'}$. We see that the gain in efficiency of computing w_k instead of s_k comes from running the Lanczos procedure using only k iterations and not attempting to compute an eigen decomposition.

Besides w_k, there is another alternative to the expensive calculation of s_k. Let $\tilde{A} = XX^T$ be applied to the Lanczos algorithm which yields the equality

$$\tilde{Q}_k^T X X^T \tilde{Q}_k = \tilde{T}_k. \tag{3.11}$$

Define

$$t_i := q^T \tilde{Q}_i \tilde{Q}_i^T X, \quad i = 1, 2, \ldots \tag{3.12}$$

It can be similarly proved that the difference between t_i and s in the right singular direction v_j of X decays at least with the same order as $T_{i-j}^{-1}(\tilde{\gamma}_j)$. This in turn means that t_i is progressively closer to s in major singular directions of X as i increases, and t_k is considered a good approximation to s_k. The most economic way of computing t_k is $((q^T \tilde{Q}_k)\tilde{Q}_k^T)X$.

The choice of whether to use w_k or t_k in replace of s_k solely depends on the relative magnitude of m and n, i.e., the shape of the matrix X. The time cost of computing w_k is $O(k(nnz+n))$, while that of computing t_k is $O(k(nnz + m))$, where nnz is the number of non-zero elements in X. It is clear that w_k is a better choice when $m \geq n$, while t_k is more appropriate when $m < n$.

3.3 Relevance analysis on the subdivisions of X. The relevance analysis on each subdivision X_i resulting from either column partitioning or row partitioning follows that we compute the similarities between the query vector q and the documents on a reduced rank subspace of X_i. Recall that for LSI, this subspace is spanned by the k largest singular vectors of X_i. In our strategies, we use the subspace spanned by the Lanczos vectors.

To be specific, for column partitioning (line 2 of Algorithm 1), the relevance scores for the documents in a subdivision X_i are first computed as the vector $w_{i,k} = q^T X_i Q_{i,k} Q_{i,k}^T$, which is then scaled by the norms of the corresponding columns of $X_i Q_{i,k} Q_{i,k}^T$. Similarly, for row partitioning (lines 3 and 4 of Algorithm 2), we first compute a vector $t_{i,k} = q_i^T \tilde{Q}_{i,k} \tilde{Q}_{i,k}^T X_i$ for each X_i, sum these $t_{i,k}$ vectors for all i, and then scale the resulting vector entries by the norms of the corresponding columns of

$$\begin{bmatrix} \tilde{Q}_{1,k}\tilde{Q}_{1,k}^T X_1 \\ \vdots \\ \tilde{Q}_{p,k}\tilde{Q}_{p,k}^T X_p \end{bmatrix}.$$

4 Experimental results

We present several experimental results that show that the proposed divide and conquer strategies are effective, at least comparable to LSI, and are far more efficient. Most of the experiments were performed in Matlab under a Linux workstation with a P4 3.20GHz CPU and 4GB memory. The only exception is that the truncated SVD of the TREC data set was computed on a machine with 16GB of memory, because Matlab was unable to handle such large data using less machine memory. It is worthwhile to note that the proposed strategies with experimented data sets can be completely fit into 4GB of memory.

4.1 Data sets. With limited data sets that have the relevance judgements available, we used the following four for experiments. Data statistics are summarized in Table 1.

Table 1: Data sets.

	MED	CRAN	NPL	TREC
# terms	7,014	3,763	7,491	138,232
# docs	1,033	1,398	11,429	528,030
# queries	30	225	93	50
ave terms/doc	52	53	20	129

MEDLINE and CRANFIELD[2]**:** These are the two early benchmark data sets for information retrieval. Their typical statistics is that *the number of distinct terms is more than the number of documents* ($m > n$).

NPL[1]**:** This data set is larger than the previous two, with a distinct property that *the number of documents is more than the number of distinct terms* ($m < n$). Including the above two, these three data sets have the term-document matrices readily available from the provided links, so we did not perform additional processing on the data.

TREC[3]**:** This large data set is popular when experiments in extensive text mining applications are performed. The whole data set consists of four document collections (*Financial Times*, *Federal Register*, *Foreign Broadcast Information Service*, and *Los Angeles Times*) from the TREC CDs 4 & 5 (copyrighted). The queries are from the *TREC-8 ad hoc* task[4][5]. Similar to NPL, the extracted term-document matrix has *more documents than distinct terms* ($m < n$). The following is the specific details of how we extracted the matrix. We used the software TMG [30] to construct the term-document matrix. The parsing process included stemming, deleting common words according to the stop-list provided by the software, and removing words with no more than 5 occurrences or with appearance in more than 100,000 documents. Also, 125 empty documents were ignored. This resulted in a term-document matrix of size 138232×528030. For the queries, only the title and description parts were extracted to construct query vectors.

4.2 Implementation specs. The weighting scheme of all the term-document matrices was *term frequency-inverse document frequency* (tf-idf). To include marginal data points in subsets during bisection, we used the following criterion:

$$\begin{cases} X_+ = \{x_i \mid v_i \geq v_{\min}/10\} \\ X_- = \{x_i \mid v_i < v_{\max}/10\} \end{cases} \qquad (4.13)$$

instead of formula (2.4b). The only exception is that for TREC we used criterion (2.5). The reason will be explained later in Section 4.4.

4.3 Column partitioning. The divide and conquer strategy with column partitioning was applied to MEDLINE and CRANFIELD, since their term-document matrices have more rows than columns. Figure 5 shows the performance of this strategy (using $p = 2$ and 4 subdivisions) compared with that of the standard LSI. The figure indicates that the accuracy obtained from divide and conquer is comparable to that of LSI, while in preprocessing the former runs an order of magnitude faster than the latter.

The accuracy is measured using the 11-point interpolated average precision, as shown in (a) and (d). More information is provided by plotting the precision-recall curves. These plots are shown using a specific k for each data set in (b) and (e). They suggest that the retrieval accuracy for divide and conquer is close to that of LSI (and is much better than that of the baseline model VSM).

Plots (c) and (f) show that divide and conquer is superior to LSI for being far more economical in time. The preprocessing of the data sets includes, for divide and conquer, subdividing the term-document matrix and computing the Lanczos vectors for each subdivision; and for LSI, computing the truncated SVD. The advantage in preprocessing speed is expected for two reasons: (1) The subdividing process involves computing only the largest singular vectors, which is inexpensive; (2) the relevance analysis on subsets is an economical alternative to the truncated SVD approach.

We also report the average query times in Table 2. The proposed strategy uses only a fraction of time more than LSI in general. In theory, the computational complexities of answering queries for both methods are the same—simply computing inner products of vectors and scaling. However, since the subdivisions in the divide and conquer strategy overlap, it is not surprising to see that there is an additional overhead. Nevertheless, this overhead is small. What is encouraging is that the query times recorded for both methods are in the same order (several or several tens of milliseconds). Hence, this extra time does not affect the overall efficiency of the proposed strategy.

Table 2: Average query time (msec).

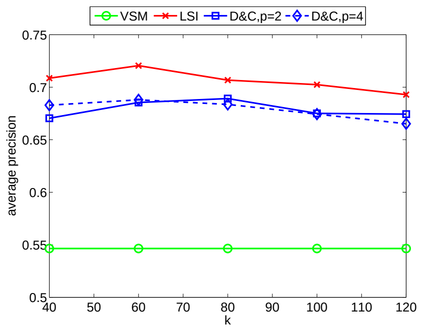
MED	LSI	D&C,2	D&C,4
$k = 40$	1.4969	2.6127	3.8976
$k = 60$	1.9818	2.6747	4.0574
$k = 80$	2.5616	2.8388	4.3142
$k = 100$	3.5161	2.9973	4.5695
$k = 120$	4.2069	3.1621	4.8515
CRAN	LSI	D&C,2	D&C,4
$k = 120$	2.0733	3.7983	5.0959
$k = 150$	2.5798	4.0787	5.5066
$k = 180$	3.0010	4.3265	5.9583
$k = 210$	3.4779	4.6118	6.2639
$k = 240$	3.9920	4.8311	6.6929

[2]`ftp://ftp.cs.cornell.edu/pub/smart/`

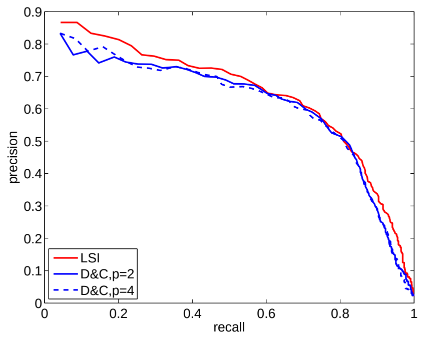
[3]`http://trec.nist.gov/data/docs_eng.html`

[4]Queries: `http://trec.nist.gov/data/topics_eng/`

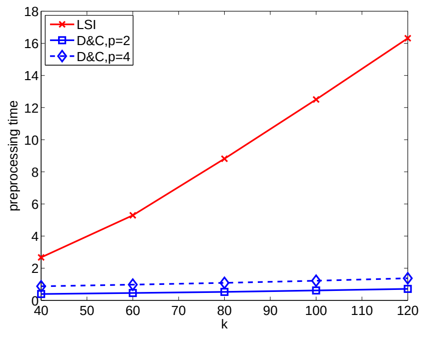
[5]Relevance: `http://trec.nist.gov/data/qrels_eng/`



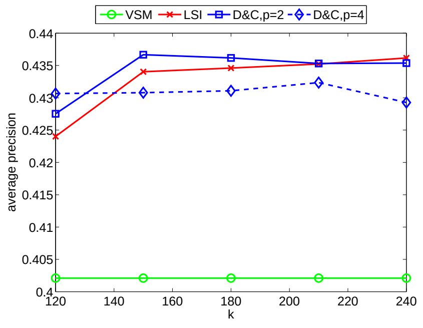
(a) MED: average precision.



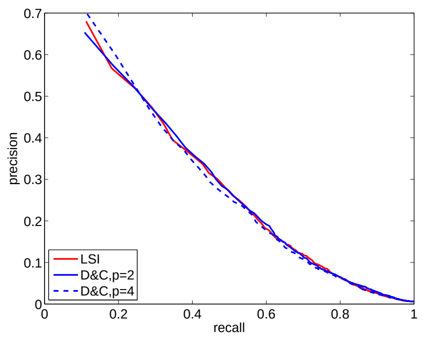
(b) MED: precision-recall ($k = 80$).



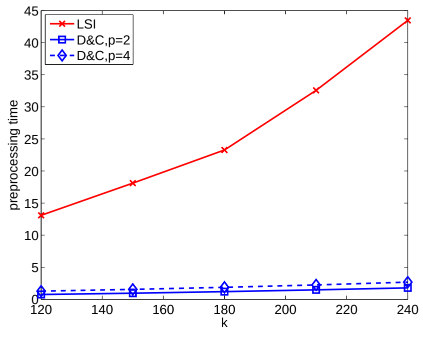
(c) MED: preprocessing time (seconds).



(d) CRAN: average precision.



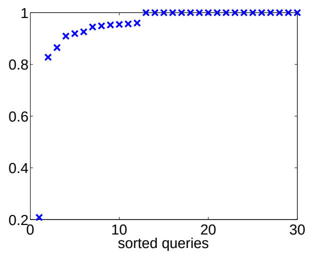
(e) CRAN: precision-recall ($k = 200$).



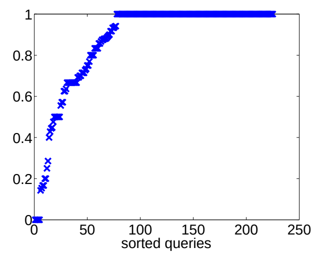
(f) CRAN: preprocessing time (seconds).

Figure 5: Performance (accuracy and time) tests on MEDLINE and CRANFIELD.

In practical scenarios, only the retrieved documents with the highest scores (i.e., top in the rank list) are of interest. This implies that for a certain query, we can perform relevance analysis only on the subset to which it belongs. This remedies the overhead of query time, at the risk of missing some relevant documents. Figure 6 shows that for most of the queries, the percentage of documents that are missed by performing analysis only on the most relevant subset is indeed low. This indicates that in applications where the query time is critical, we may reduce the amount of relevance analysis by focusing on only one subset for each query. This way the query response of our divide and conquer strategy will not be slower than that of LSI.



(a) MED



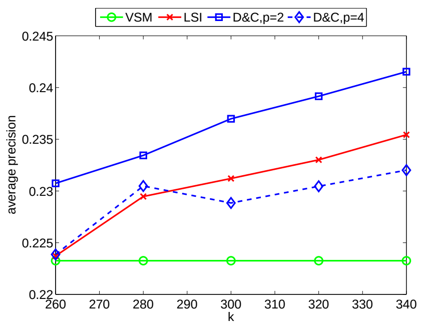
(b) CRAN

Figure 6: Percentage of relevant documents that are in the subset where a certain query belongs. (using $p = 4$ subdivisions)

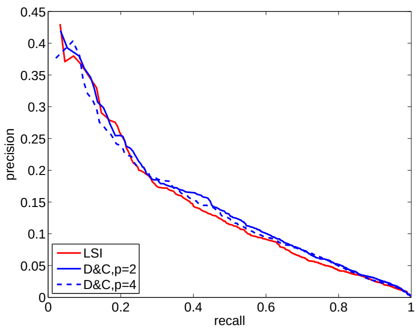
4.4 Row partitioning. The divide and conquer strategy with row partitioning was applied to NPL and TREC, since their term-document matrices have more columns than rows. Figure 7 shows the performance of this strategy when applied to NPL. Similar to Figure 5 (the column partitioning strategy), Figure 7 indicates that divide and conquer with row partitioning yields comparable accuracy to LSI, while it is an order of magnitude more efficient for preprocessing. The query times are again in the order of milliseconds; see Table 3.

We also performed tests on TREC. Since we have no access to a fair environment (which required as much as 16GB working memory and no interruptions from other users) for time comparisons, we tested only the accuracy. Figure 8 plots the precision-recall curves. The data set was partitioned into $p = 4$ and 8 divisions. As shown in the figure, divide and conquer greatly improves over LSI, especially at low recalls, which represent the earliest appearance of relevant documents.

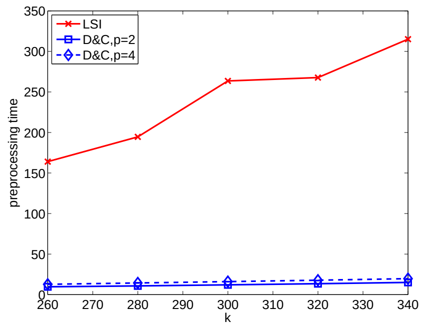
Besides the above encouraging result, two facts are



(a) NPL: average precision.



(b) NPL: precision-recall ($k = 300$).



(c) NPL: preprocessing time (seconds).

Figure 7: Performance (accuracy and time) tests on NPL.

Table 3: Average query time (msec).

NPL	LSI	D&C,2	D&C,4
$k = 260$	17.9422	24.6421	22.9736
$k = 260$	18.7971	25.2407	24.4162
$k = 260$	20.2458	26.3537	24.0412
$k = 260$	21.6721	26.7478	27.1805
$k = 260$	23.3515	28.7238	28.1093

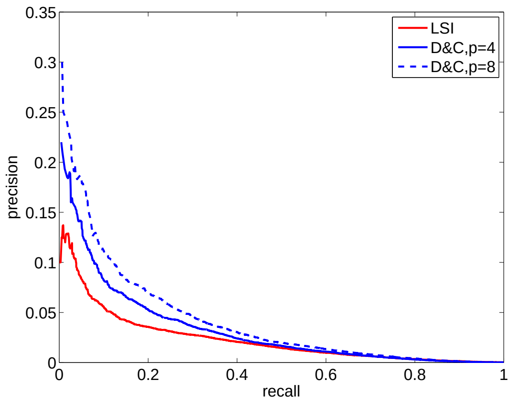


Figure 8: TREC: precision-recall ($k = 300$).

worth being mentioned here. The first is that in contrast with other experiments which used criterion (4.13) to bisect the data set, for TREC we used criterion (2.5), that is, dividing the data set into perfectly balanced subsets. This follows from an observation of the very skewed distribution of the terms in TREC. Figure 9 illustrates this phenomenon. The entries of the largest right singular vector v of the matrix $X^T - ce^T$ (c.f. Section 2.1) are proportional to the distances between data points (terms in this case) and the dividing hyperplane. As shown in Figure 9, only a little more than 10% of the terms are on one side of the hyperplane, and the remaining ones, on the other side, are located very close to this plane. If we had divided the data set using criterion (4.13), it would have resulted in very imbalanced subsets. This not only affected the effectiveness of the parallelism of the proposed strategy, but also yielded poor results. This unusual skewness of the distribution is by itself an interesting phenomenon worth further investigation.

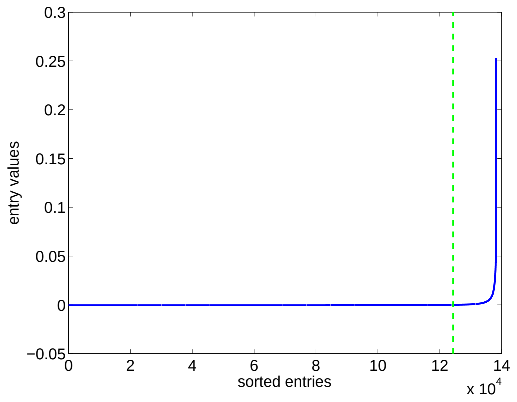


Figure 9: Entry values of the largest right singular vector v of the matrix $X^T - ce^T$ where X is the term-document matrix of TREC. The dashed vertical line separates the negative entries from the positive entries.

The second important fact is that the overall accuracy is not high. While our main effort is to design strategies that can compete with LSI, we note that for this example LSI did not perform as well as expected. This may be due to the weighting scheme of the term-document matrix or the extraction method/-tool we used. Indeed, different weighting schemes, as well as what terms are filtered out in the matrix, may have a significant impact on the accuracy of LSI (see [20] for some discussions). What this paper presents are two strategies that compete with LSI regardless of the term extraction process and the weighting scheme. The latter factor is itself a research topic that has been widely addressed in the literature.

5 Conclusions and future work

Two divide and conquer strategies are proposed to effectively retrieve relevant documents for text mining problems, along with an efficient technique to use as an alternative to the classical truncated SVD approach for relevance analysis. Experimental results show that these strategies yield comparable retrieval accuracy to the standard LSI, and are an order of magnitude faster. In addition, these strategies are easily amenable to parallel implementations. Because of their inherent parallelism, they can be deployed for solving much larger problems than be handled by standard LSI.

Several aspects are worth future investigation following this work. As mentioned in Section 2.3, reordering techniques of sparse matrices can be exploited to devise more divide and conquer strategies. In Section 4.4, the skewed distribution of terms implies that spectral bisection may not be effective in clustering for certain data sets, or at least modifications of the technique are needed. Also, the question of "what is the optimal number of subdivisions" is as difficult to answer as "how many clusters a data set has". The answer is certainly application and data set specific. Finally, it would be interesting to see how term extraction methods and weighting schemes can impact the effectiveness of the proposed strategies.

References

[1] C. Aykanat, A. Pinar, and U. urek. Permuting sparse rectangular matrices into block-diagonal form. Technical report, Computer Engineering Department, Bilkent University, 2002.

[2] M. Berry and M. Browne. *Understanding search engines: Mathematical Modeling and Text Retrieval*. SIAM, 2nd edition, 2005.

[3] M. Berry, S. Dumais, and G. O' Brien. Using linear algebra for intelligent information retrieval. *SIAM Rev.*, 37(4):573–595, 1995.

[4] M. Berry and R. Fierro. Low-rank orthogonal decompositions for information retrieval applications. *Numer. Lin. Alg. Appl.*, 1:1–27, 1996.

[5] Michael W. Berry. Large scale sparse singular value computations. *International Journal of Supercomputer Applications*, 6(1):13–49, 1992.

[6] Daniel Boley. Principal direction divisive partitioning. *Data Mining and Knowledge Discovery*, 2(4):325–344, 1998.

[7] Matthew Brand. Fast low-rank modifications of the thin singular value decomposition. *Linear Algebra Appl.*, 415(1):20–30, 2006.

[8] A. Brandt. Multi-level adaptive solutions to boundary value problems. *Mathematics of Computation*, 31:333–390, 1977.

[9] W. L. Briggsa, V. E. Henson, and S. F. Mc Cormick. *A multigrid tutorial*. SIAM, 2nd edition, 2000.

[10] Jie Chen and Yousef Saad. Lanczos vectors versus singular vectors for effective dimension reduction. *IEEE Trans. Knowl. Data Eng.*, in submission.

[11] E. Chisholm and T. Kolda. New term weighting formulas for the vector space method in information retrieval. Technical report, Oak Ridge National Laboratory, 1999.

[12] S. Deerwester, S. Dumais, G. Furnas, T. Landauer, and R. Harshman. Indexing by latent semantic analysis. *J. Soc. Inf. Sci.*, 41:391–407, 1990.

[13] Carl Eckart and Gale Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211–218, 1936.

[14] J. Erhel, F. Guyomarc, and Y. Saad. Least-squares polynomial filters for ill-conditioned linear systems. Technical report, University of Minnesota Supercomputing Institute, 2001.

[15] Haw-Ren Fang and Yousef Saad. Farthest centroids divisive clustering. In *The Seventh International Conference on Machine Learning and Applications (ICMLA'08)*, 2008.

[16] Jing Gao and Jun Zhang. Clustered SVD strategies in latent semantic indexing. *Information Processing & Management*, 41(5):1051–1063, 2005.

[17] G. H. Golub and C. F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, 3rd edition, 1996.

[18] W. Hackbusch. *Multi-Grid Methods and Applications*, volume 4 of *Springer Series in Computational Mathematics*. Springer-Verlag, Berlin, 1985.

[19] Bruce Hendrickson and Tamara G. Kolda. Partitioning rectangular and structurally unsymmetric sparse matrices for parallel processing. *SIAM J. Sci. Comput.*, 21(6):2048–2072, 2000.

[20] Parry Husbands, Horst Simon, and Chris Ding. Term norm distribution and its effects on latent semantic indexing. *Information Processing and Management*, 41(4):777–787, 2005.

[21] F. Juhász and K. Mályusz. *Problems of cluster analysis from the viewpoint of numerical analysis*, volume 22 of *Colloquia Mathematica Societatis Janos Bolyai*. North-Holland, Amsterdam, 1980.

[22] E. Kokiopoulou and Y. Saad. Polynomial filtering in latent semantic indexing for information retrieval. In *ACM-SIGIR Conference on research and development in information retrieval*, 2004.

[23] T. Kolda and D. O' Leary. A semi-discrete matrix decomposition for latent semantic indexing in information retrieval. *ACM Trans. Inf. Syst.*, 16(4):322–346, 1998.

[24] C. Lanczos. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. *J. Res. Nat. Bur. Stand.*, 45:255–282, 1950.

[25] Y. Saad. *Numerical Methods for Large Eigenvalue Problems*. Halstead Press, New York, 1992.

[26] Yousef Saad. Filtered conjugate residual-type algorithms with applications. *SIAM J. Matrix Anal. Appl.*, 28(3):845–870, August 2006.

[27] Jane E. Tougas and Raymond J. Spiteri. Updating the partial singular value decomposition in latent semantic indexing. *Comput. Statist. Data Anal.*, 52(1):174–183, 2007.

[28] D. Tritchler, S. Fallah, , and J. Beyene. A spectral clustering method for microarray data. *Comput. Statist. Data Anal.*, 49:63–76, 2005.

[29] D. I. Witter and M. W. Berry. Downdating the latent semantic indexing model for conceptual information retrieval. *The Computer J.*, 41(8):589–601, 1998.

[30] D. Zeimpekis and E. Gallopoulos. *TMG: A MATLAB toolbox for generating term document matrices from text collections*, pages 187–210. Springer, Berlin, 2006.

[31] Hongyuan Zha and Horst D. Simon. On updating problems in latent semantic indexing. *SIAM J. Sci. Comput.*, 21(2):782–791, 1999.

[32] Hongyuan Zha and Zhenyue Zhang. Matrices with low-rank-plus-shift structure: Partial SVD and latent semantic indexing. *SIAM J. Matrix Anal. Appl.*, 21(2):522–536, 1999.