Divide and Conquer Strategies for Effective Information Retrieval*

Jie Chen and Yousef Saad
Department of Computer Science and Engineering
University of Minnesota at Twin Cities
Minneapolis, MN 55455
{jchen, saad}@cs.umn.edu

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 is 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 applied to LSI, with the goal of alleviating these difficulties. These strategies process a data set by dividing it in subsets and conquering the LSI results on each subset. Since each sub-problem resulting from the divide and conquer strategy 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 of the LSI analysis of the subsets, we employ an approximation technique that is based on the Lanczos algorithm. This technique is far more efficient than the truncated SVD, while its accuracy is comparable. Experimental results confirm that the proposed divide and conquer strategies are effective for information retrieval problems.

Categories and Subject Descriptors

H.3 [**Information Search and Retrieval**]: Retrieval models; H.3 [**Content Analysis and Indexing**]: Indexing methods

General Terms

Algorithms

Keywords

Information retrieval, latent semantic indexing, divide and conquer, multilevel, graph partitioning

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

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

$$s = q^T X \tag{1}$$

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

The vector space 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. Let X have

the SVD

$$X = U\Sigma V^T, \tag{2}$$

with the truncated rank-k version:

$$Y_k = U_k\Sigma_k V_k^T, \tag{3}$$

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, 15]. Up to scalings, the vector produced by LSI is given by

$$s_k = q^T Y_k, \tag{4}$$

which replaces the expression (1) of the vector space model.

Current implementations of LSI mainly rely on matrix decompositions (see e.g., [4, 20]), 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 some update of the SVD, and this is a not an easy task. Much research has been devoted to the problem of updating the (truncated) SVD [24, 26, 7, 22], but these methods have not exploited the sparsity of X. In addition the resulting SVD will lose accuracy after frequent updates.

To bypass the truncated SVD computation, Kokiopoulou and Saad [19] introduced polynomial filtering techniques, and Chen and Saad [10] proposed approximations using the Lanczos vectors. These methods all efficiently compute a sequence of vectors that progressively approximate the vector s_k defined in (4), without resorting the expensive singular value decomposition.

While a number of papers have addressed the issue of computational cost of LSI, few considered the *divide and conquer* paradigm as a means to lessen the computational burden. In this paper we propose two divide and conquer strategies with a primary goal of reducing cost at the expense of a minimal loss in accuracy. Divide and conquer is particularly attractive for very large problems when performing the SVD is difficult. In the proposed strategies, we recursively divide the data set using spectral partitioning techniques, and separately perform LSI analysis on each subset. (This analysis can be done via the standard SVD approach, or through an efficient method based on the Lanczos algorithm.) 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. These advantages will be demonstrated in the experiment section on two different types of data sets: either the number of distinct terms exceeds the number of documents, or the reverse.

The rest of the paper is organized as follows. Section 2 presents the two divide and conquer strategies. Section 3 discusses the efficient alternative to the truncated SVD for the LSI 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, 16], 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 in which the analysis of similarities between the query and the documents is performed. Because the sets are smaller, the analysis, be it done by the SVD or Lanczos ([10, 5]), will tend to be much less expensive. The main motivation for divide and conquer strategies is the simple fact that it is usually much cheaper to solve k eigenproblems of size $n \times n$ than one of size $(kn) \times (kn)$. Though we are computing only partial spectra and the problem is that of the SVD instead of the eigen decomposition, the general argument remains valid. 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 there are many more columns than rows and the row set in the opposite situation. However, *a little cost analysis along with experimentation show that the opposite is computationally more appealing.*

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 partitioning*, a simple-to-implement technique whose effectiveness has been well documented in the literature, see, e.g., [6, 23, 18, 14].

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

$$X_+ = \{x_i \mid u^T(x_i - c) \geq 0\} \quad \text{and} \quad X_- = \{x_i \mid u^T(x_i - c) < 0\}. \tag{5a}$$

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

$$X_+ = \{x_i \mid v_i \geq 0\} \quad \text{and} \quad X_- = \{x_i \mid v_i < 0\}, \tag{5b}$$

where v is *the largest right singular vector* of $\bar{X}$. If it is preferred that the sizes of the clusters are balanced, an al-

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

ternative is to replace the above criterion by

$$X_+ = \{x_i \mid v_i \geq \mathrm{m}(v)\} \quad \text{and} \quad X_- = \{x_i \mid v_i < \mathrm{m}(v)\}, \quad (6)$$

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

These ideas were discussed in [6, 23, 18] and a few improvements to the technique were considered in [14]. In particular, the largest left/right singular vectors can be computed via the Lanczos algorithm in a very cheap cost 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 by looking at the signs of v_i. In this way we are implicitly partitioning the *hypergraph* which is canonically associated with the matrix X when the hyperedges are defined from the columns of X.

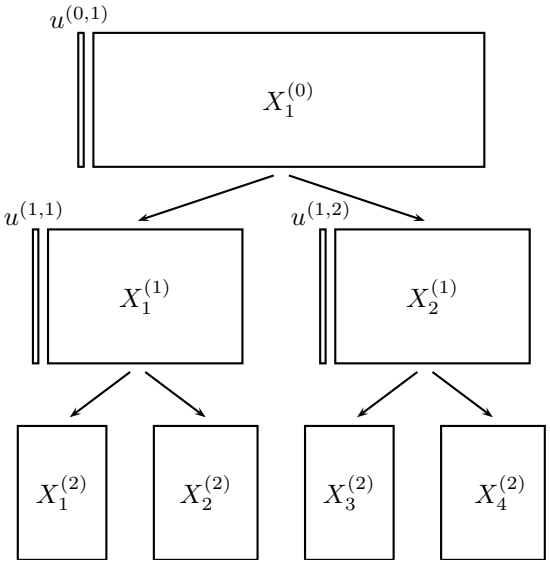


Figure 1: A 2-level division of a term-document matrix X. The vectors $u^{(i,j)}$ shown for levels 0 and 1 are perpendicular to the separating hyperplanes associated with the nodes. Note that this figure is not drawn to scale—column partitioning is better applied to tall and thin matrices.

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.

At each node in the hierarchy tree, a vector of size 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.

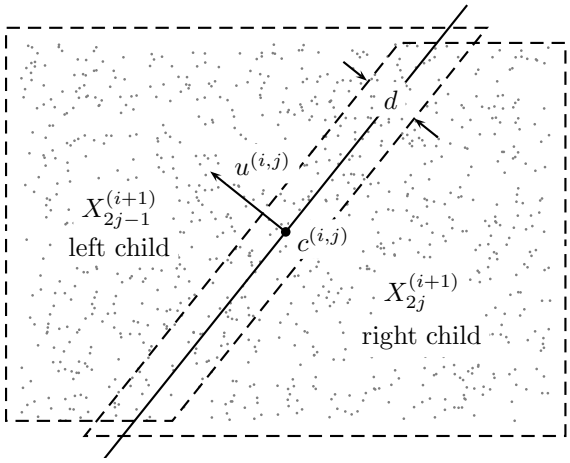


Figure 2: Illustration of partitioning a node $X_j^{(i)}$ into two children $X_{2j-1}^{(i+1)}$ and $X_{2j}^{(i+1)}$. The vector $u^{(i,j)}$ always points towards the left child. The two dashed frames indicate the actual configuration of the two subsets, which share a margin of width d around the hyperplane.

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

For 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 LSI analysis can be performed on each subset and similarity scores are produced. In Section 3 we will see a more efficient way to perform this analysis instead of computing the truncated SVDs for the subsets. In this section, let us temporally disregard how the analysis is performed and simply consider that similarity scores are available.

If a document appears in only one leaf node, the similarity 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 analysis scores from these nodes to be the similarity 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 boarder of several clusters. The question "how relevant this document is to the query" may yield different answers if the analysis is performed separately 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. (Hierarchical bisection.)
2: For a given query q, compute similarity scores between q and those in the leaf nodes of the hierarchy. (By truncated SVD, or by Lanczos described in Section 3.)
3: The similarity 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 section when $m \leq n$. The partitioning approach is identical to that of the previous section with the simple exception that it is applied to the transpose of X.

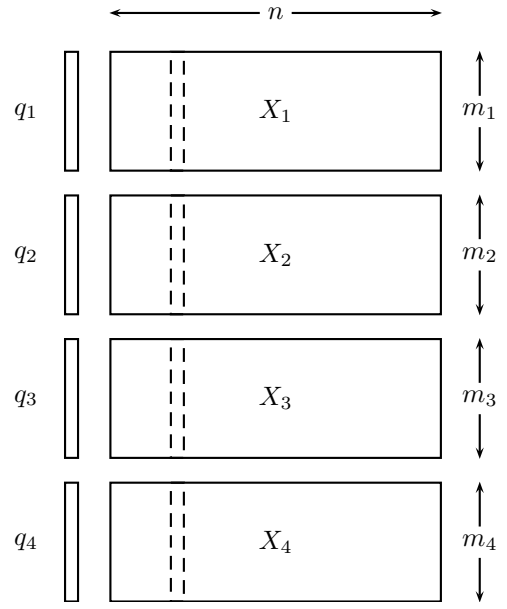


Figure 3: A term-wise subdivision of a term-document matrix X into 4 subsets. The query q is subdivided accordingly. Note that this figure is not drawn to scale—row partitioning is better applied to wide and short matrices.

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

$$X = \begin{bmatrix} X_1 \\ \vdots \\ X_p \end{bmatrix} \quad \text{and} \quad q = \begin{bmatrix} q_1 \\ \vdots \\ q_p \end{bmatrix}. \tag{7}$$

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 applied to 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 subdivisions (leaf nodes) instead of using the (i, j) tuple to index all the subsets in the hierarchy as in the previous section.

2.2.1 Query response

The question now is how to compute the similarities between a query and the columns of the original matrix X. If we had just computed the inner products of the partial columns of X and 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 LSI analysis 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

$$X_i = U_i \Sigma_i V_i^T \tag{8}$$

with the truncated rank-k version

$$Y_{i,k} = U_{i,k} \Sigma_{i,k} V_{i,k}^T. \tag{9}$$

Then the similarity scores for all the documents are

$$s'_k = \sum_{i=1}^{p} q_i^T Y_{i,k} \tag{10}$$

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 (10) is very similar to (4), 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.

Algorithm 2 D&C-LSI: Row Partitioning

1: Recursively bisect the rows of X, resulting in submatrices X_1, X_2, ..., 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 (10).
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 (10) with (4), this divide and conquer strategy essentially performs an analysis with the matrix $\hat{X}$ replacing X in a 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 information by

performing LSI 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 1. *Let* $\text{best}_k(\cdot)$ *denote the best rank-k approximation of a matrix. Consider*

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

where $X_1 \in \mathbb{R}^{m_1 \times n}$ *and* $X_2 \in \mathbb{R}^{m_2 \times n}$ *with* $m_1 + m_2 \leq n$. *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 [27]. (Also see the remark thereafter.) Hence the proof is omitted here. Note that the theorem is restricted to situations when $m \leq n$, since otherwise XX^T is singular.

Theorem 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 (7), 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 [27] 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., [17, 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. ANALYSIS USING LANCZOS APPROXIMATION

As mentioned earlier, the LSI analysis (formula (4)) is expensive due to the computation of the truncated SVD. In the divide and conquer strategies previously described, this analysis can be replaced by an efficient technique called *Lanczos approximation* [10]. This technique essentially computes a good approximation to the vector s_k defined in (4).

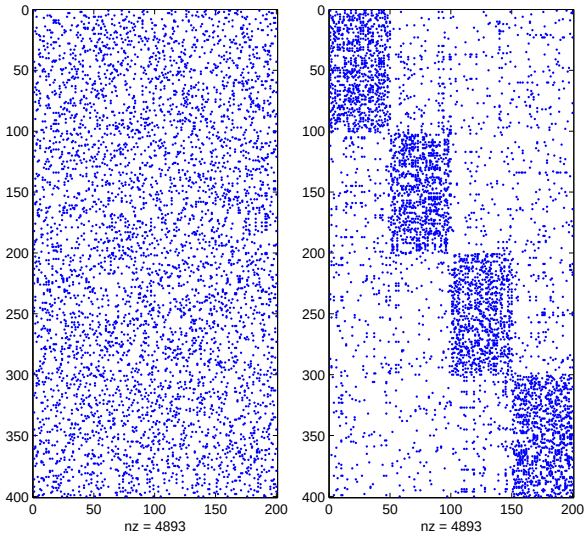


Figure 4: A small rectangular sparse matrix before and after reordering in block diagonal form.

The effectiveness of this technique applied to the whole data set X has been demonstrated in the previous report [10]. In this paper we can apply this technique to each subdivision of X. In the sequel we first briefly review the symmetric Lanczos algorithm, which is later used to develop the approximation technique.

3.1 The symmetric Lanczos algorithm

Given a symmetric matrix $A \in \mathbb{R}^{n \times n}$ and an initial unit vector 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 resulted 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 [15, 21].

3.2 Lanczos approximation

A good approximation of the vector s_k defined in (4) can be efficiently obtained by exploiting the Lanczos procedure reviewed in the previous section. Let $A = X^T X$ be applied to the Lanczos algorithm which yields equality

$$Q_k^T X^T X Q_k = T_k. \tag{11}$$

By defining

$$w_i := q^T X Q_i Q_i^T, \quad i = 1, 2, \ldots \tag{12}$$

i.e., letting w_i be the projection of $s = q^T X$ onto the subspace $\mathrm{ran}(Q_i)$, 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 good approximation of 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 cheaper than the computation of s_k. In fact, computing s_k requires computing the truncated SVD of X. A typical way of computing the k largest singular components of X is to go through the Lanczos process as in (11) 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 Lanczos using only k iterations and not pursuing an extra eigen decomposition.

The computation of an approximation vector to s_k can be further explored by considering the relative size of m and n—shape of the matrix. When $m \geq n$, the matrix $A = X^T X$ is of size $n \times n$, and w_k defined in (12) is the appropriate one to approximate s_k. However, when $m < n$, we consider another approximation. Let $\tilde{A} = XX^T$ be applied to the Lanczos algorithm which yields equality

$$\tilde{Q}_k^T X X^T \tilde{Q}_k = \tilde{T}_k. \tag{13}$$

Define

$$t_i := q^T \tilde{Q}_i \tilde{Q}_i^T X, \quad i = 1, 2, \ldots \tag{14}$$

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 good approximation of s_k. The most economic way of computing t_k is $((q^T \tilde{Q}_k)\tilde{Q}_k^T)X$.

The choice of using w_k or t_k to approximate s_k solely depends on the relative size of m and n. The cost of computing w_k, including the Lanczos process, is $O(k(nnz + n))$, where nnz is the number of non-zero elements in X. On the other hand, the cost of computing t_k is $O(k(nnz + m))$. It is clear that w_k is a better choice when $m \geq n$, while t_k is more appropriate when $m < n$.

In summary, in order to efficiently implement the divide and conquer strategies, the scores vector computed in line 2 of Algorithm 1 (column-partitioning version) is replaced by w_k defined in (12), and the scores vector computed in line 3 of Algorithm 2 (row-partitioning version) is replaced by t_k defined in (14). Entries of w_k (or t_k) are then scaled with the norms of the corresponding columns of $XQ_kQ_k^T$ (or $\tilde{Q}_k\tilde{Q}_k^T X$).

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 7 under a Linux workstation with two P4 3.00GHz CPUs 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.

4.1 Data sets

Four data sets were used in the experiments:

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.* The reason to use these two data sets is that they are relatively small and thus convenient for numerous prototyping tests. Statistics is shown in Table 1.

NPL[2].

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.* Statistics is also shown in Table 1.

TREC[3].

This is a very large data set which is popularly used for experiments in serious text mining applications. Similar to NPL, the term-document matrix extracted from this data set has *more documents than distinct terms.* Specifically, 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]. We used the software TMG [25] 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.

Table 1: Data sets statistics.

	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

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

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

used the following criterion:

$$X_+ = \{x_i \mid v_i \geq v_{\min}/10\} \quad \text{and} \quad X_- = \{x_i \mid v_i < v_{\max}/10\} \tag{15}$$

instead of formula (5b). The only exception is that for TREC we used criterion (6). 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 orders 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).

The preprocessing of the data set(s) 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. As shown in (c) and (f), divide and conquer is superior to LSI—while being far more econimical. This is expected for two reasons: (1) The subdividing process involves computing only the largest singular vector(s), which is very inexpensive; (2) the Lanczos approximation is an economical alternative to the truncated SVD for this particular task. We also report the average query times in Table 2. The computational complexities for both methods are the same. Since the subdivisions in the divide and conquer strategy overlap, it is not surprising to see that it takes a little more time to answer queries using this strategy. However, since the times recorded are in the same order (milliseconds), this extra time does not impact the overall efficiency of the divide and conquer strategy.

Table 2: Average query time (msec).

MED	LSI	D&C,2	D&C,4
$k = 40$	4.0897	9.6917	14.3992
$k = 60$	6.2982	9.5132	15.2941
$k = 80$	8.3144	9.8150	15.8666
$k = 100$	9.9056	9.9923	16.2019
$k = 120$	12.3475	10.1446	16.4957
CRAN	LSI	D&C,2	D&C,4
$k = 120$	7.3504	13.2534	17.7076
$k = 150$	9.0561	13.5225	18.5583
$k = 180$	10.8678	14.0904	19.3488
$k = 210$	12.4254	14.7360	19.8805
$k = 240$	14.0872	15.3105	20.8624

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 6 shows the performance of this strategy when applied to NPL. Similar to Figure 5 (the column partitioning strategy), Figure 6 indicates that divide and conquer with row partitioning yields comparable accuracy to LSI, while it is orders of magnitude more efficient for preprocessing. The query times are again in the order of milliseconds; see Table 3.

Table 3: Average query time (msec).

NPL	LSI	D&C,2	D&C,4
$k = 260$	33.7209	76.2982	77.9623
$k = 280$	36.9476	78.7518	81.1645
$k = 300$	39.0302	80.6269	82.8548
$k = 320$	41.4048	82.7120	84.7012
$k = 340$	44.5710	85.7431	87.9671

Figure 7 plots the precision-recall curves for the TREC data set, which is partitioned into $p = 4$ and 8 divisions in the divide and conquer technique. As shown in the figure, divide and conquer improves over LSI, especially at low recalls, which represent the earliest appearance of relevant documents.

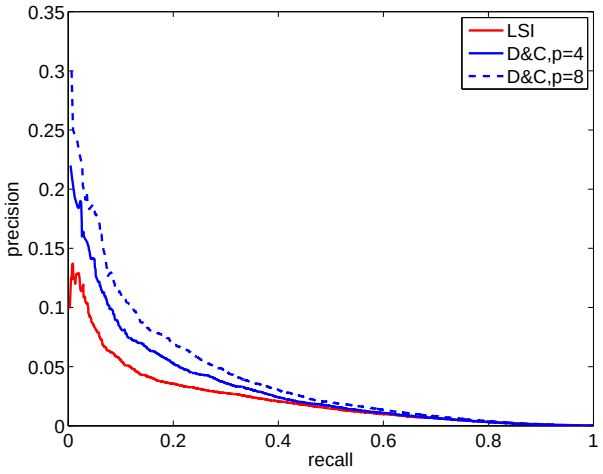
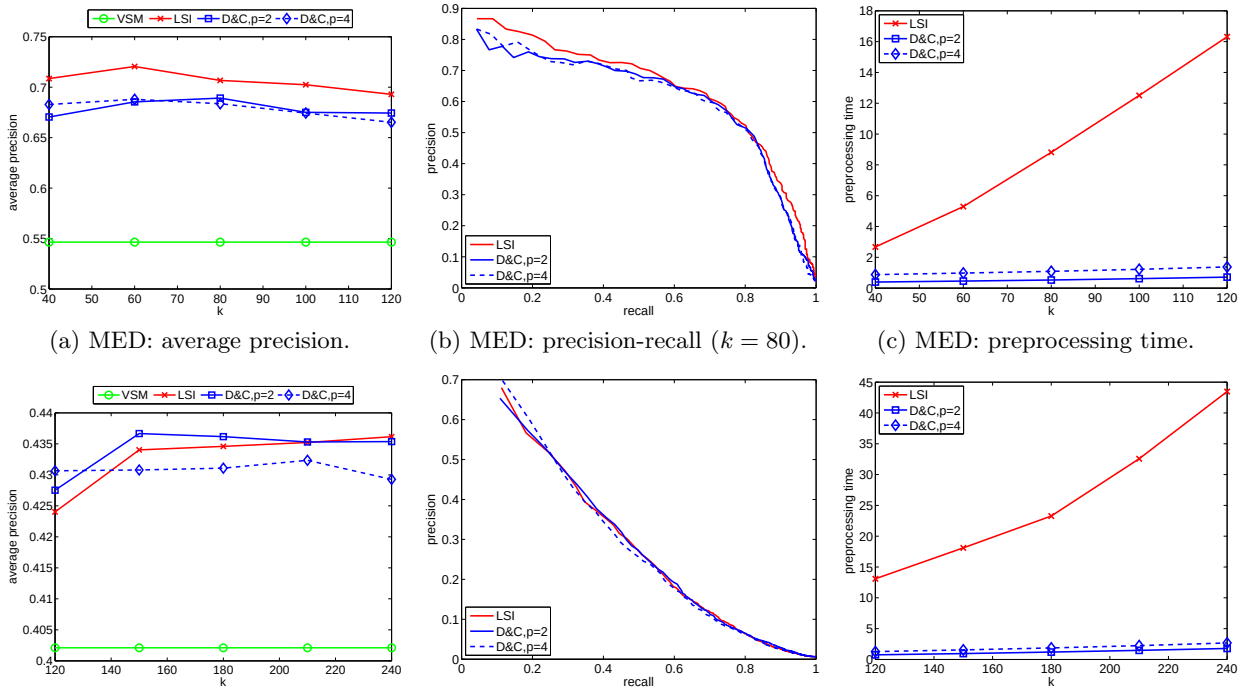


Figure 7: TREC: precision-recall ($k = 300$).

Besides the above encouraging result, two facts are worth being mentioned here. The first is that in contrast with other experiments which used criterion (15) to bisect the data set, for TREC we used criterion (6), 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 8 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 8, only a little more than 10% of all 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 (15), 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.

The second important fact is that the overall accuracy is not high. While our main effort is to design strategies that



(a) MED: average precision. (b) MED: precision-recall ($k = 80$). (c) MED: preprocessing time.

(d) CRAN: average precision. (e) CRAN: precision-recall ($k = 200$). (f) CRAN: preprocessing time.

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

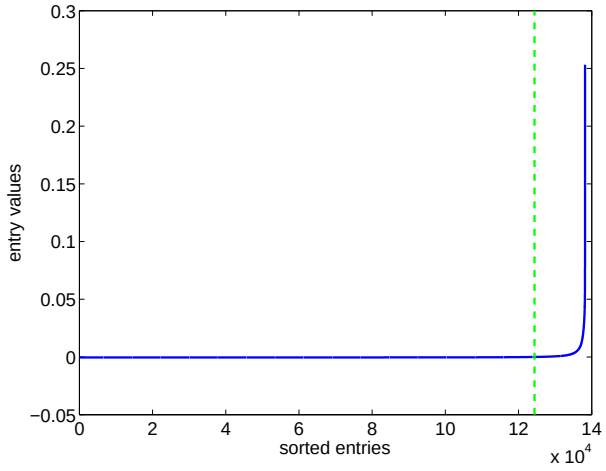


Figure 8: Entry values of the largest right singular vector v of the matrix $X^T - ce^T$ where X is the term-document matrix of TREC. The dashed vertical line separates the negative entries from the positive entries.

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 indexing method/tool we used.

5. CONCLUSIONS AND FUTURE WORK

Two divide and conquer strategies were proposed to effectively retrieve relevant documents for text mining problems, along with efficient techniques to use as alternatives to the classical truncated SVD approach of LSI. Experimental results show that these strategies yield comparable retrieval accuracy to the standard LSI, and are orders 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 partitioning techniques may not be effective in clustering the data for certain data sets, or at least modifications of the techniques are needed. Finally, it would be interesting to see how indexing methods and weighting schemes can impact the effectiveness of the divide and conquer strategies.

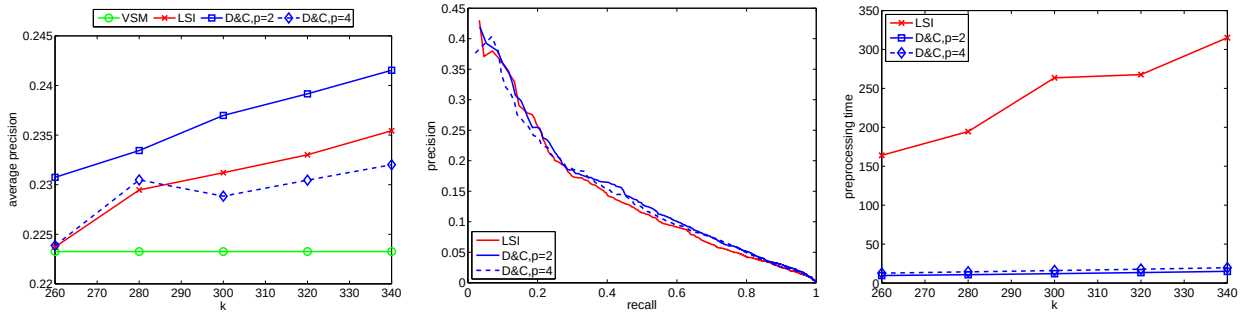
6. REFERENCES

[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 etrieval.* 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.



(a) NPL: average precision. (b) NPL: precision-recall ($k = 300$). (c) NPL: preprocessing time.

Figure 6: Performance (accuracy and time) tests on NPL.

[5] K. Blom and A. Ruhe. A krylov subspace method for information retrieval. *SIAM J. Matrix Anal. Appl.*, 26(2):566–582, 2005.

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

[7] M. 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] J. Chen and Y. Saad. Filtered matrix-vector products via the Lanczos algorithm with applications to dimension reduction. Technical Report UMSI 2008/2, University of Minnesota Supercomputing Institute, 2008.

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

[14] H.-R. Fang and Y. Saad. Farthest centroids divisive partitioning. Technical Report UMSI 2008/5, University of Minnesota Supercomputing Institute, 2008.

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

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

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

[18] 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.

[19] 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.

[20] 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.

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

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

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

[24] 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.

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

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

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