International Journal of Computer Applications (0975 8887)
Volume 99 - No. 10, August 2014

Multistep Sparse Approximation Technology in Information Retrieval

Chi Shen
Kentucky State University
400 E. Main Street
Frankfort, KY 40601

Wen Li
Lexmark International, Inc.
740 W New Circle Rd
Lexington, KY 40511

Mike F. Unuakhalu
Kentucky State University
400 E. Main Street
Frankfort, KY 40601

ABSTRACT

With large sets of text documents increasing rapidly, being able to efficiently utilize this vast volume of new information and service resource presents challenges to computational scientists. Text documents are usually modeled as a term-document matrix which has high dimensional and space vectors. To reduce the high dimensions, one of the various dimensionality reduction methods, concept decomposition, has been developed by some researchers. This method is based on document clustering techniques and least-square matrix approximation to approximate the matrix of vectors. However the numerical computation is expensive, as an inverse of a dense matrix formed by the concept vector matrix is required. In this paper we presented a class of multistep spare matrix strategies for concept decomposition matrix approximation. In this approach, a series of simple sparse matrices are used to approximate the decompositions. Our numerical experiments on both small and large datasets show the advantage of such an approach in terms of storage costs and query time compared with the least-squares based approach while maintaining comparable retrieval quality.

General Terms:

Computer Information Retrieval

Keywords:

Term-document matrix, Concept decomposition matrix, Sparsity pattern, Multistep, Sparse matrix approximation

1. INTRODUCTION

The text retrieval method using Latent Semantic Indexing (LSI) with the low-rank Singular Value Decomposition (SVD) of the term-document matrix has been intensively studied for years [1, 9, 16]. Although the LSI method has empirical success, it suffers from the lack of interpretation for the low-rank approximation [6] and may be less effective for large heterogeneous text collections [9, 10]. A method introduced by [5] is an improvement in that direction. In this approach, a large amount of dataset is first partitioned into a set of disjoint clusters. The centroids of clusters or also known as concept decomposition is computed and normalized as the concept vector of that cluster for lowering the rank of the term-document matrix [6]. Gao and Zhang [7] have indicated that retrieval accuracy from the concept decomposition can be comparable to that from LSI. However, the numerical computation based on the straightforward implementation of the concept matrix decomposition is expensive, as an inverse of the normal matrix (which is likely to be a dense matrix) formed by the concept vector matrix is required during the query procedure. The dense data structure of the concept decomposition matrix poses a huge challenge for both disk and memory spaces of conventional computers. Some researchers [2, 7, 11, 18] have recently applied sparsification techniques to reduce computational cost and memory space of concept decomposition method. In [7] it proposed a sparsification strategy by dropping small entries during the concept matrix inverse computation. They have demonstrated that the sparsified concept decomposition technique may improve the performance in terms of retrieval accuracy and storage cost for some cases compared with SVD. The approach used in [11] is quite different from [7]. The LSI input matrices have been sparsified by identifying and removing some values from submatrices that are used in LSI methods. Shen and Williams [18] utilized the knowledge of both preconditioning techniques and the computational theorems to compute a sparse matrix to approximate the projection matrix directly. The key part in their approach is the sparsity pattern strategies. Different strategy patterns may result in quite different performance. This method based on different sparsity pattern strategies greatly reduces memory cost and query time while maintaining close retrieval accuracy to that of the straightforward implementation of the concept matrix decomposition method.

In an effort to improve the retrieval accuracy while maintaining a sparse query matrix, the authors of this paper developed a class of multistep sparse matrix approximation strategies to compute the matrix associated with the concept matrix decomposition. These computations are related to weighted least squares optimization and approximate pseudo matrix inversion. In this method, a sequence of sparse non-square matrices are computed at each step. The sum of these sparse matrices along with the concept matrix are used to approximate the original term-document matrix. The more steps that are used, the more accurate approximate matrix we will have. Our approach to sparsification is different from the ones in [7, 2, 18]. We utilize the knowledge of sparse inverse technologies developed in the preconditioning field [4, 17] and apply it to concept matrix decomposition approximation for non-square sparse matrices. Our experimental results show that these approaches improve the retrieval accuracy compared to the approaches in [18] while using

less CPU time and memory space required to compute the decomposition matrix and perform document retrieval.
This paper is organized as follows. Concept decomposition techniques and least squares matrix approximation based retrieval procedures are discussed in Section 2 and Section 3. These are followed by a detailed discussion of the multistep sparse matrix approximation based strategies in Section 4. Numerical experiments are presented in Section 5. We summarize this paper in Section 6.

2. DOCUMENT CLUSTERING AND CONCEPT VECTORS

In information retrieval, text documents are modeled as a term-document matrix whose rows are the terms and columns are document vectors. Suppose we are given the set of document vectors $A_{m\times n} = [a_1, a_2, \ldots, a_n]$, where $a_j, (j = 1, 2, \ldots, n)$ is the jth document vector of m dimension in the collection. Usually, the term-document matrices are high dimensional as each data set may contain thousands of words and are very sparse, with 1-5% or less terms in each document.
We first partition the documents into k disjoint clusters $\{\pi_j\}_{j=1}^k$ by using k-means or other clustering algorithms such that

$$\bigcup_{j=1}^{k} \pi_j = \{a_1, a_2, \ldots, a_n\} \text{ and } \pi_j \cap \pi_i = \phi \text{ if } j \neq i.$$

For each fixed j, $1 \leq j \leq k$, the centroid vector of each cluster is defined as

$$\widetilde{c_j} = \frac{1}{n_j} \sum_{a_i \in \pi_j} a_i,$$

where $n_j = |\pi_j|$ is the number of documents in cluster π_j. The centroid vectors are normalized in following

$$c_j = \frac{\widetilde{c_j}}{\| \widetilde{c_j} \|}, \ \ j = 1, 2, \ldots, k.$$

An intuitive definition of the clusters is that, if $a_i \in \pi_j$, then

$$a_i^T c_j > a_i^T c_l \quad \text{for} \quad l = 1, 2, \ldots, k, \quad l \neq j,$$

i.e., documents in π_j are closer to its centroid than to the other centroids. If the clustering is not good enough, a new set of centroid vectors for the new partitioning is computed. Each document will be reassigned to a new cluster based on the new computed centroid vector. This process can be repeated a few times to a stopping threshold value of ne. If each cluster is compact enough, the centroid vector may represent the abstract concept of the cluster. So they are also called *concept vectors* [5]. The *concept matrix* can be defined as an $m \times k$ matrix such that,

$$C = [c_1, c_2, \ldots, c_k].$$

The concept matrix C that has rank $m \times k$ is still a sparse matrix, as each column c_j is sparse. This is because c_j is a group of documents in A with similar key words, i.e. the columns of A that have similar number of non-zeros at almost the same fill-in positions. Due to A's sparsity, c_j is sparse too. The degree of sparsity of C is inversely proportional to the number of clusters generated. If we assume that the concept vectors are linearly independent, the concept matrix has rank k.
For any partitioning of the document vectors, we can define the corresponding concept decomposition $\tilde{A}$ of the term-document matrix A as the least squares approximation of A onto the column space of the concept matrix C. We write the concept decomposition as an $m \times n$ matrix

$$\tilde{A} = C\tilde{M},$$

where $\tilde{M}$ is a $k \times n$ matrix that is to be determined by solving the following least squares problem

$$\tilde{M} = \arg\min_M \|A - CM\|_F^2. \tag{1}$$

Here the norm $\| \cdot \|_F$ is the Frobenius norm of a matrix. It is well-known that problem (1) has a closed-form solution, i.e.,

$$\tilde{M} = (C^T C)^{-1} C^T A. \tag{2}$$

3. RETRIEVAL PROCEDURE AND APPROXIMATE LEAST SQUARES BASED STRATEGIES

As it had been described in [8, 18], given a query vector q, the retrieval with the concept projection matrix (CPM) $\tilde{A}$ can be computed as

$$r^T = q^T \tilde{A} = q^T C(C^T C)^{-1} C^T A, \tag{3}$$

where r^T is the ranking vector. After sorting the entries of r^T in a descending order, we have a ranking list of documents which are related to the query (listed from the most relevant to the least relevant). The retrieval procedure in Eq. (3) is not efficient in terms of numerical computation and storage when the number of cluster k is large as it needs to compute $(C^T C)^{-1}$, which is likely to be a dense matrix.
To make the numerical computation procedure and storage cost more competitive, a sparse matrix approximation technique based on the static and dynamic sparsity pattern strategies has been studied in [18]. In this method, a sparse matrix M is computed to solve the least squares problem (1) approximately. That is to find a sparse matrix M such that the functional $f(M) = \|A - C\,M\|_F^2$ is minimized, or more precisely, *approximately minimized*. In [18], static and dynamic sparsity pattern strategies had been discussed. In their numerical experiments, both of the approaches greatly reduced the computational costs and storage costs. Compared to the straightforward implementation of concept matrix decomposition method CPM, both strategies use less than 5% memory storage of the CPM approach while achieving comparable retrieval accuracy.
Obviously there is much room to improve the sparsity pattern strategies. To have more accurate sparse approximate matrix, it is natural to think of increasing the sparsity ratio for M. However, the CPU time used for matrix M construction increases as well. Dynamic sparsity pattern strategies can compute better sparse approximate matrix, given a certain sparsity ratio for M. However, they are usually expensive. In this paper we propose a multistep matrix approximation strategy based on static sparsity pattern approach. At each step, we compute a series of sparse, or cheap matrix $M_i, i = 1, 2, \ldots$ at each step. Then the sum of these sparse matrices is used to approximate the matrix M. This method takes the advantage of cheaper construction at each step and a denser matrix we will obtain at last.

4. MULTISTEP STATIC SPARSITY PATTERN APPROACH (MSSP)

In our problem in information retrieval, all we want is to find a $sparse$ matrix M such that

$$f(M) = \min_{M \in \mathcal{G}} \|A - CM\|_F^2, \quad (4)$$

is minimized with certain constraints on M . This problem is similar to that in the equation in preconditioning field:

$$f(M) = \min_{M \in \mathcal{G}} \|I - AM\|_F^2, \quad (5)$$

Several sparse approximate inverse methods for solving above equation have been developed in [4, 17]. We will use these technologies for our problem.

4.1 Computational procedure

For a moment, we suppose that a sparsity pattern set $\mathcal{G}$ for M is given somehow, the minimization problem (4) is decoupled into n independent subproblems as

$$\|A-CM\|_F^2 = \sum_{j=1}^{n} \|(A-CM)e_j\|_2^2 = \sum_{j=1}^{n} \|a_j - Cm_j\|_2^2, \quad (6)$$

where a_j and m_j are the jth column of the matrices A and M, respectively. (e_j is the jth unit vector.) It follows that the minimization problem (6) is equivalent to minimizing the individual functions

$$\|Cm_j - a_j\|_2, \qquad j = 1, 2, \ldots, n, \quad (7)$$

with certain restrictions placed on the sparsity pattern of m_j. In other words, each column of M can be computed independently. This certainly opens the possibility for parallel implementation. Since we assume the sparsity pattern of m_j (and M) is given, i.e., a few, say n_2, entries of m_j at certain locations are allowed to be nonzero, the rest of the entries of m_j are forced to be zero. Denote the n_2 nonzero entries of m_j by $\bar{m}_j$ and the n_2 columns of C corresponding to $\bar{m}_j$ by C_j. Since C is sparse, the submatrix C_j has many rows that are identically zero [4, 17]. After removing the zero rows of C_j, we have a reduced matrix $\bar{C}_j$ with n_1 rows. The individual minimization problem (7) is reduced to a much smaller least squares problem of order $n_1 \times n_2$

$$\|\bar{C}_j \bar{m}_j - \bar{a}_j\|_2, \qquad j = 1, 2, \ldots, n, \quad (8)$$

in which $\bar{a}_j$ consists of the entries of a_j corresponding to the remaining columns of C_j. We note that the matrix $\bar{C}_j$ is now a very small rectangular matrix. It has full rank if the matrix C does.

There are a variety of methods available to solve the small least squares problem (8). Assume that $\bar{C}_j$ has full rank. Since $\bar{C}_j$ is small, the easiest way is probably to perform a QR factorization on $\bar{C}_j$ as

$$\bar{C}_j = Q_j \begin{pmatrix} R_j \\ 0 \end{pmatrix}, \quad (9)$$

where R_j is a nonsingular upper triangular $n_2 \times n_2$ matrix. Q_j is an $n_1 \times n_1$ orthogonal matrix, such that $Q_j^{-1} = Q_j^T$. The least squares problem (8) is solved by first computing $\bar{c}_j = Q_j^T \bar{a}_j$ and then obtaining the solution as $\bar{m}_j = R_j^{-1} \bar{c}_j(1 : n_2)$. In this way, $\bar{m}_j$ can be computed for each $j = 1, 2, \ldots, n$, independently. This yields an approximate decomposition matrix M, which minimizes $\|CM - A\|_F$ for the given sparsity pattern.

4.2 Multistep approach

From the above discussion, we can see that our computational procedure is quite similar to that in [4, 17]. However it is much more difficult than that in the preconditioning field. This is because of the fact that A is a $m \times n$ matrix, not a square matrix. The dimensions of the matrix C and those of M do not match. That is C is a $m \times k$ matrix and M is a $k \times n$ matrix. This multistep knowledge from the preconditioning field can not be directly applied to our application. We approach the problem for non-square matrix minimization in another way.

Suppose a simple and cheap sparsity pattern is chosen for M_1. M_1 is computed approximately for the equation

$$CM = A \quad (10)$$

by using our modified sparse approximate inverse techniques developed in [18]. If CM_1 is not close enough to A, we can compute another approximate sparse matrix M_2 for the equation

$$CM = A - CM_1 \quad (11)$$

Based on Eq.(11), $CM_2 \approx A - CM_1$, we sum the two matrices M_1 and M_2, such that $C(M_1+M_2) \approx A$. Generally, $C(M_1+M_2)$ should be closer to A than CM_1 does. If $C(M_1 + M_2)$ is not good enough for the matrix A, we can compute the third sparse matrix M_3 for the equation

$$CM = A - C(M_1 + M_2) \quad (12)$$

such that $C(M_1 + M_2 + M_3) \approx A$. This procedure can be repeated for a few times to obtain a sequence of sparse matrices $M_1, M_2, ..., M_l$ such that $C(M_1 + M_2 + ... + M_l) \approx A$. As [17] pointed out a few sparse matrices $M_l M_{l-1} \ldots M_1$ for matrix approximate inverse may be capable of holding more information than a single matrix M can. In our application, if each M_i is good in some sense, we expect that

$$\lim_{l->\infty} C(M_1 + M_2 + ... + M_l) = A$$

In general, the more accurate is matrix M, the more dense it is. Hence the computational cost may also increase. In our approach, since we sum the matrices at each step, it shouldn't have many new fill-ins. For example, if the number of nonzeros of the matrix M_1 is n_1, and the number of nonzeros of the matrix M_2 is n_2, the number of nonzeros of the matrix $M_1 + M_2$ would be less than $n_1 + n_2$. If M_1 and M_2 have the same sparsity patterns, there would be no new fill-ins. To avoid too few new fill-ins at each step, we choose different sparsity patterns for M_1 and M_2.

We first define the sparsity pattern for M_1. Note that the concept matrix C describes the relationship between the term vectors and the concept vectors. If a term is related to a concept vector, this relationship may be maintained in the approximate decomposition matrix in some sense. The strategy based on the sparsity pattern of C and entry values of both C and A is described below.

From the equation $CM_1 = A$, we know that $c_i m_j^{(1)} = a_{ij}$, where c_i is the ith row of C, $m_j^{(1)}$ is the jth column of M_1. The sparsity pattern of $m_j^{(1)}$ is given in this way: If a_{ij} is the largest entry in the jth column of A, the sparsity pattern of $m_j^{(1)}$ is the same as that of c_i. Here we use small matrices to illustrate our ideas. Suppose the three matrices are $C_{4\times3}$, $M_{1_{3\times5}}$ and $A_{4\times5}$. The pattern of $CM_1 = A$ is depicted by the Eq (13).

$$\begin{pmatrix} x & 0 & x \\ 0 & x & 0 \\ x & x & 0 \\ 0 & x & 0 \end{pmatrix}_{4\times3} \quad \begin{pmatrix} - & - & - & - & - \\ - & - & - & - & - \\ - & - & - & - & - \end{pmatrix}_{3\times5}$$

$$= \begin{pmatrix} 0 & x & 0 & x & 0 \\ 0 & 0 & 0 & x & x \\ x & x & 0 & 0 & 0 \\ x & 0 & x & 0 & 0 \end{pmatrix}_{4\times 5} \quad (13)$$

Here, "x" denotes nonzero entry, "-" denotes undefined pattern. We determine the sparsity pattern of M_1 column by column. First, find the largest entry in each column of A, suppose they are a_{31}, a_{12}, a_{43}, a_{14}, and a_{25} in Eq (13). Then the sparsity pattern of $m_1^{(1)}$, is the same as that of c_3 and the sparsity pattern of $m_2^{(1)}$ is the same as that of c_1, $m_3^{(1)}$ has same sparsity pattern of c_4 and $m_4^{(1)}$ is the same as that of c_1. Finally we have the sparsity pattern of M_1 like this: $\begin{pmatrix} x & x & 0 & x & 0 \\ x & 0 & x & 0 & x \\ 0 & x & 0 & x & 0 \end{pmatrix}$.

For the sparsity pattern of matrix M_2 at the second step, we consider the equation $CM_2 = A - CM_1$. We first sort non-zero entries in each column of $A - CM_1$. Then we go through all the entries from the largest to the smallest. Then the first ne number of largest entries will be used as new fill-ins for M_2. If the positions of these selected entries have been used in matrix M_1, these entries will continue to be used in the sparsity pattern in M_2, but they are not counted as new fill-ins. So other larger entries will be selected to reach the total ne number of new fill-ins. A parameter ne is used to control the number of new fill-ins for each column of M_i at the ith step. So in general the matrix M_i is denser than M_{i-1}. We had conducted some numerical tests for this strategy. With a few new fill-ins for the matrix M_2, the 2-norm residual of $\|A - C(M_1 + M_2)\|_2$ reduces much more than that of $\|A - C(M_1 + M_2)\|_2$ with no more new fill-ins in M_2. That means the denser matrix $M_1 + M_2$ may hold more information. This multistep procedure may be repeated for a few times as needed.

5. NUMERICAL EXPERIMENTS

In this section, we conducted some experiments to compare the performance of our proposed algorithms MSSP and the single-step approximate sparse approach SSP and CPM method that is based on the straightforward implementation by solving Eq.(2). Three small text databases: CRAN,MED and CISI and two large databases: LISA and NPL are used to validate our approach. The information about the five databases are given in Table 1.

Precision-recall curves are often used to evaluate the performance of the algorithms in information retrieval [15]. In the following experiments, we first test the precision-recall curves for different approaches. We average the precision of all queries at fixed recall values as $10\%, 20\%, \ldots, 90\%$. Then we compare their storage costs by listing the number of nonzeros of the approximate matrix M obtained in Eq. (2). The CPU time required for the approximate matrix computation and query procedure is also reported for the tests over small datasets. The sparse matrix multistep computations are based on ParaSails developed in [4] by using C. This computation procedure and matrix inverse for concept project matrix method (CPM) were carried out in IBM Power/Intel(Xeon) hybrid system at University of Kentucky. The computations for query procedure were carried out in a SunOS system by using matlab.

In all the following tests, "$nstep$" denotes the number of steps used in sparse matrix M construction; "k" is the number of clusters used in k-mean algorithm; "nnz" represents the number of nonzeros for the matrix M constructed with different methods. "ne", a threshold value, is the number of new fill-ins in each column of M at each step. As it was discussed in Section 4, the sparsity pattern of each column of matrix M depends on the number of the largest entries selected in the corresponding column of A. The parameter "ne" means the first ne number of largest entries in each column of A that have not been used in the previous step will be selected . The larger the number of ne, the more dense is the matrix of M. This usually leads to more accurate matrix M, and more computational and memory costs. This is because the storage space for the MSSP is primarily determined by the number of largest entries considered at each step. To balance the accuracy and the costs, in our tests we choose $ne = 2$ for multistep approach. In order to compare our multistep approach with the single-step SSP more precisely, we increase the density of SSP by setting $ne = 4$. We denote it as "SSP_4".

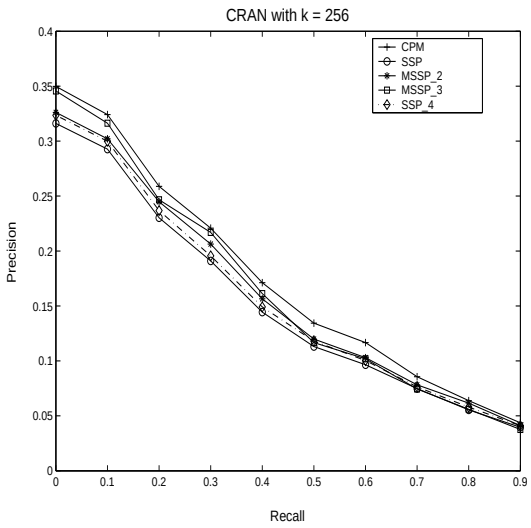

Fig. 1. Precision-Recall Results for CRAN with $k = 256$

We first test the precision-recall curves for the CRAN database with $k = 256$. As expected in Fig. 1, CPM has best query results. This is because CPM matrix is constructed by directly solving the Eq. (2). The resulting matrix is very dense(we will show this in the later tests) and hence more accurate than that of the other approximate approaches. From the Fig. 1 we can see the multistep algorithm MSSP with $nstep = 2, 3$ which does outperform the single-step approach SSP. This is partially because of more dense matrix constructed in multistep approach. However more dense matrix doesn't always generate more accurate matrix. Table 2 lists the numbers of nonzeros of M constructed with different approaches. Looking into our two-step approach MSSP that utilizes 2 largest entries for each column at each level and the single-step SSP that using 4 largest entries from the Table 2, we find that the SSP_4 constructs more dense matrix than MSSP. But in Fig. 1 we observe that 2-step MSSP has better retrieval accuracy than that of single-step SSP_4. That means with the same amount of memory cost, multistep approach usually construct more accurate matrix. This encouraging finding is what allowed us to intensify our efforts in developing multistep methods. We also make comparison of a 3-step MSSP and CPM. In Fig. 1 and Table 2 we found that the precision-recall curve for MSSP with $nstep = 3$ is very close to that of the CPM approach while 3-step MSSP uses much less memory than CPM. MSSP uses only 24% of CPM memory while achieving 90% of accuracy that CPM has. However the computational cost will increase as well. We could see this in the following test for CRAN database with $k = 500$.

Table 1. The information of databases

Database	Matrix size	Number of queries
CISI	$5,609 \times 1,460$	112
CRAN	$4,612 \times 1,398$	225
MED	$5,831 \times 1,033$	30
LISA	$17,482 \times 6,004$	35
NPL	$11,380 \times 11,429$	93

Table 2. Storage costs for CRAN database, k = 256

Methods	ne	$nstep$	nnz
SSP	2	1	45,719
MSSP	2	2	75,986
SSP_4	4	1	93,878
MSSP	2	3	86,761
CPM	-	-	357,888

The test results listed in Table 3 are from testing the same database with the number of clusters: $k = 500$. Table 3 shows that the 2-step MSSP has better query results while using less memory space than SSP_4 approach. The data listed in the last two rows of Table 3 are the query results and nonzeros for the matrix M of 3-step MSSP and CPM. In this case, 3-step MSSP approach uses only 18% of CPM storages to achieve 93% retrieval accuracy of CPM. From this experiment, we also observe that the computational cost and query time have been greatly reduced.

Fig. 2 gives both query results and storage costs for testing the database MED with $k = 200$. From the left panel of Fig. 2, MSSP with $nstep = 3$ has really close query accuracy to that of CPM approach. From the right panel of Fig. 2, we see that 3-step MSSP consumes only 8% of storage space of CPM. This is because the dense matrix of CPM may have many noise entries. Our sparsification algorithm is designed to drop those unnecessary entries and keep very few entries that may hold more important information in the matrix. However it is difficult to predetermine those nonzero entries or sparsity pattern before solving the problem. This is still a challenging work in scientific field. We have tried to drop some small entries from CPM matrix. We found out that the query precisions are also degraded and it still uses more than 10 times of storage than the 2-step MSSP matrix with comparable query accuracy. This experiment again assures that a larger number of steps leads to better performance in terms of memory costs and competitive query precision.

We give the test for the dataset CISI. The query results, memory costs are given in Fig. 3. In this test, the multistep approach MSSP has slightly better performance than single step approach SSP and SSP_4 in terms of query precisions, query time and memory costs. Again, the higher values of $nstep$, or ne lead to more work, more fill-in in M matrix , and usually more accurate matrix. From the case reported in previous tests, we think that 2 or 3 steps MSSP is a good compromise between reasonable computational cost and comparable performance in terms of query precision.

Below are the tests over two large datasets LISA and NPL. We first test the precision-recall curves and storage costs for the LISA database with $k = 250$.

Fig. 4 shows that our multistep algorithms have very close precision results to the one of CPM algorithm. However from the Fig. 5, we see the multistep algorithms that use much less memory spaces, about 50%, than that of CPM.

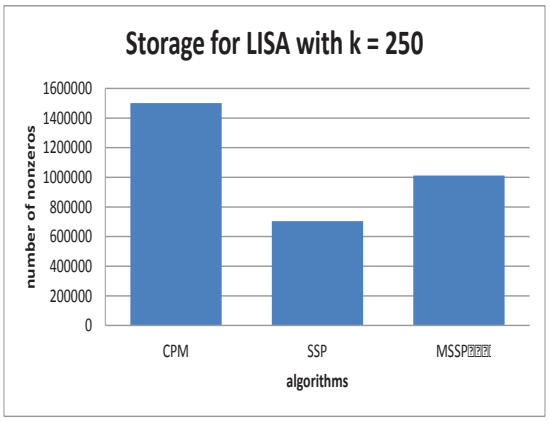

Fig. 5. Storage costs for LISA with $k = 250$

From above results, we also observe that compared with the single-step strategy, the multistep strategy can produce higher precision result. This is because the matrix constructed by MSSP may hold more information than the one constructed by SSP. However, as we stated earlier, denser approximate matrix may not always lead to better performance. Besides, the threshold value of ne is very essential during the construction of the M, so we use different threshold values to adjust the information held in M.

The test on LISA dataset has shown the advantages of the multistep algorithms over CPM method. When we performed the NPL dataset, the results were mixed.

The above two figures in Fig. 6 are the precision-recall curves of NPL when it has been grouped into 500 clusters. A close observation of the two curves shows a much better result in left panel over right panel of Fig. 6 in spite of its higher nonzero entries. See the storage costs in Fig. 5. Therefore, from this example, we can further conclude that in MSSP the denser approximate matrix M may not achieve better performance due to potential noisy data that are captured in the denser matrix.

However, for NPL dataset, MSSP is not as robust as for LISA for different numbers of clusters. We tested NPL with $k = 800$ in Fig. 8 and it shows that our MSSP is not comparable with CPM. Compared with the methods used in [13, 12, 14, 3], the size of NPL preprocessed in our way is much larger and the data matrix has the feature of $m < n$, i.e. the document's number is more than the term's number. We think that MSSP does not work well due to some specific features of NPL.

6. CONCLUSION

This study has explored an issue of multistep approximate matrix computation methods for the concept matrix decomposition. In many respects, the 3 step MSSP and CPM produced very similar query results except for the storage costs which are less for

Table 3. Precision-Recall Results for CRAN database, k = 500

Methods	ne	$nstep$	nnz	recall 10%	40%	80%
SSP	2	1	49,974	0.3377	0.1897	0.0603
MSSP	2	2	91,044	0.3629	0.2181	0.0702
SSP_4	4	1	107,480	0.3580	0.2007	0.0649
MSSP	2	3	124,559	0.3831	0.2202	0.0724
CPM	–	–	699,000	0.4086	0.2390	0.0759

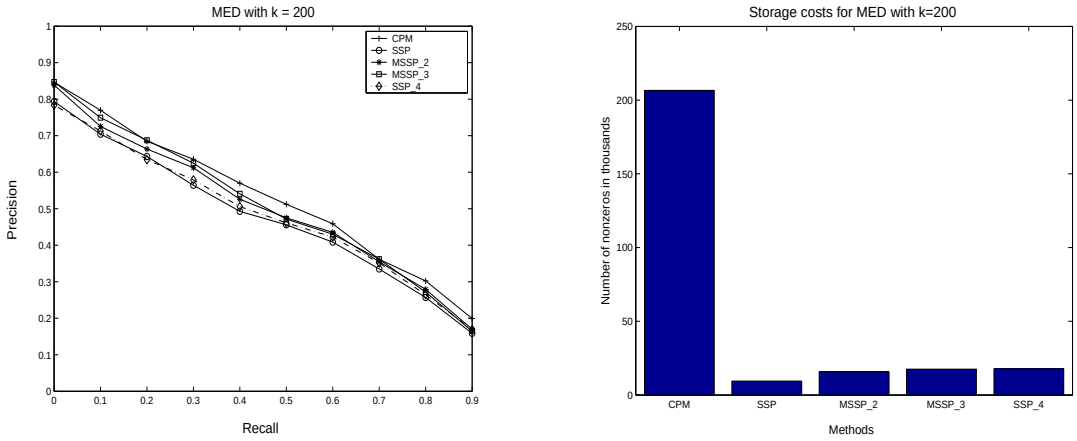

Fig. 2. Left panel: Query results for MED with k=200. Right panel: Storage costs for MED with k = 200.

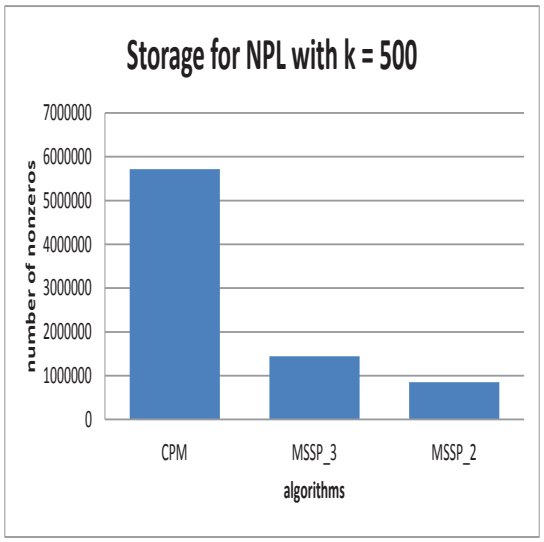

Fig. 7. Storage costs for NPL with $k = 500$

MSSP than CPM method. To compare our multistep MSSP with the single-step approach SSP under the constraint of the same amount of memory cost, we allow more nonzero entries (this can be controlled by adjusting the parameter ne) at the first step of the SSP sparse matrix construction phase. Our numerical experiments have demonstrated that SSP can compute more dense matrix, but may not be more accurate compared with the multistep approach. Additionally, MSSP is cheaper in its matrix computation phase as evidenced at each step of MSSP which resulted in a more sparse matrix. As was indicated in [17], computing a series of small memory cost (sparse) matrices usually cheaper than computing a high memory cost (dense) matrix.

Although in our study, the performance of MSSP on NPL is not as good as it is on LISA and other small datasets, the advantages of MSSP in saving CPU memory space and computational costs have still been demonstrated. The best precision result of MSSP on NPL can be comparable with that in [14].

The multistep matrix approximation approach developed in this paper has provided a new avenue to approximate the concept decomposition matrix more cheaply and fast. If the storage cost and query time are the bottle-necks during the query procedure, this approach looks more attractive. There is much room for improvement. In our next step, the algorithm parallelism will be our first target. Due to the nature of the algorithm described in the paper, it will certainly open the door for the parallelism during the matrix approximation process. We then test it on some real larger corpora, like OHSUMED and some collections from TREC to better understand the performance of this approach. We also hope this multistep sparse matrix approximation approach may be applied to the general nonsquare sparse matrix approximation techniques for functional minimization $f(M) = \min \|A - CM\|_F^2$.

7. REFERENCES

[1] M.Berry and M.Browne. Understanding Search Engines: Mathematical modeling and text retrieval. *SIAM*,1999

[2] C.M.Chen, N.Stoffel, M.Post, C.Basu, D. Bassu, and C.Behrens. Telcordia LSI engine: Implementation and scalability issues. In *Proceedings of the Eleventh International Workshop on Research Issues in Data Engineering (RIDE 2001)*, Heidelberg, Germany, April 2001.

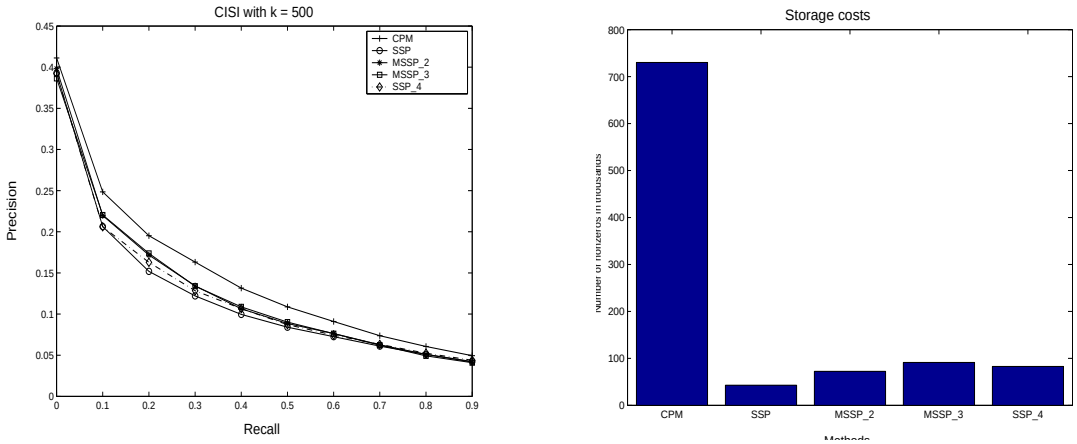

Fig. 3. Left panel: Query results for CISI with k=500. Right panel: The Storage costs.

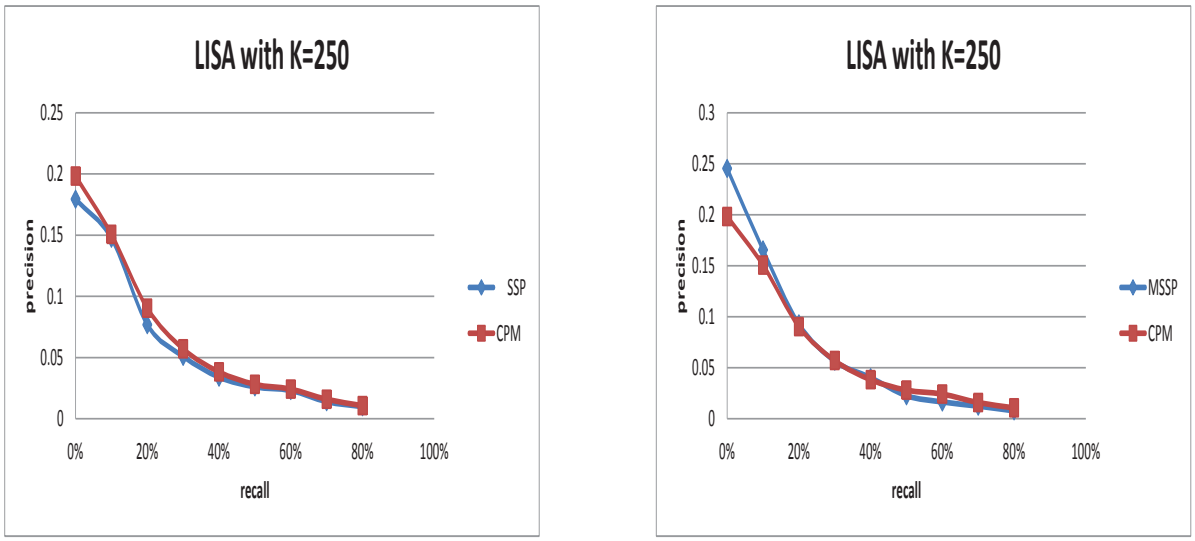

Fig. 4. Precision-recall for LISA with $k = 250$ Left panel: CPM and SSP. Right panel: CPM and MSSP with 3 steps.

[3] J. Chen and Y. Saad. Divide and Conquer Strategies for Effective Information Retrieval, *Proceedings of the Ninth SIAM International Conference on Data Mining (SDM)*, 2009.

[4] E. Chow. A priori sparsity patterns for parallel sparse approximate inverse preconditioners. *SIAM J. Sci. Comput.*, 21(5):1804–1822, 2000.

[5] I. S. Dhillon and D. S. Modha. Concept decompositions for large sparse text data using clustering. *Machine Learning*, 42(1):143–175, 2001.

[6] J.Dobsa and B.J. Basic. Concept decomposition by fuzzy k-means algorithm, In *Proceedings of IEEE Web Intelligence Conference (WI2003)*, pp. 684-688, October 2003

[7] J. Gao and J. Zhang. Text retrieval using sparsified concept decomposition matrix. Book Chapter of Computational and Information Science, Springer Berlin/Heidelberg, vol. 3314, pp. 523 – 529, 2005

[8] J. Gao and J. Zhang. Clustered SVD strategies in latent semantic indexing. in *Information Processing and Management*, 41(5):1051-1063, 2005.

[9] P. Husbands, H. Simon, and C. Ding. On the use of singular value decomposition for text retrieval. In M. W. Berry, editor, *Computational Information Retrieval*, pages 145–156. SIAM, Philadelphia, PA, 2001.

[10] E. R. Jessup and J. H. Martin. Taking a new look at the latent semantic analysis approach to information retrieval. In *Proceedings of the SIAM Workshop on Computational Information Retrieval*, Raleigh, NC, 2000.

[11] A.Kontostathis, W.M.Pottenger, and B.D.Davison. Identifcation of critical values in Latent Semantic Indexing (LSI). *Book chapter of Foundations of Data Mining and Knowledge Discovery*, pp.333-346, 2005.

[12] A. Kontostathis and W. M. Pottenger. A framework for understanding Latent Semantic Indexing(LSI) performance. *In-*

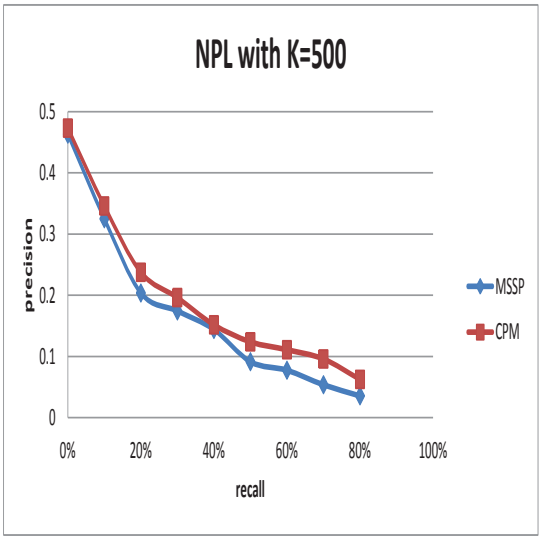

Fig. 6. Precision-recall for NPL with $k = 500$ Left panel: CPM and MSSP with 2 steps. Right panel: CPM and MSSP with 3 steps.

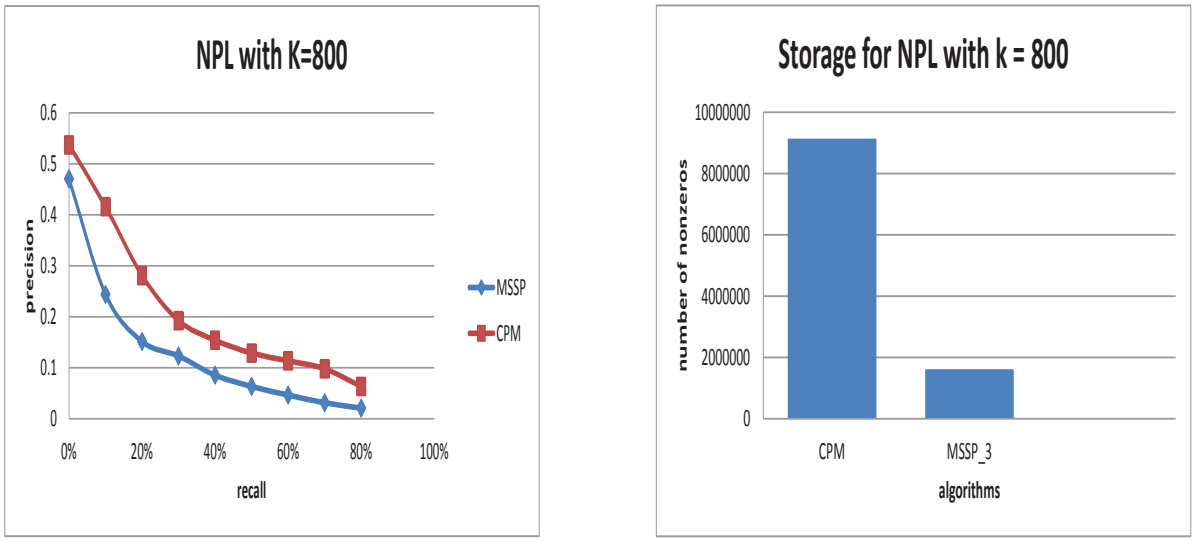

Fig. 8. Left panel:Precision-recall for NPL with $k = 800$ and 3 steps. Right panel: Storage costs for NPL with $k = 800$.

formation Processing and Management , 42(1):56-73, 2006.

[13] A. Kontostathis, W.M. Pottenger, and B.D.Davison. Identification of critical values in Latent Semantic Indexing(LSI). *In Foundations of Data Mining and Knowledge Discovery* pp. 333-346, 2005.

[14] A. Kontostathis. Essential Dimensions of Latent Semantic Indexing (LSI). *40th Annual Hawaii International Conference on System Sciences*, pp. 73, 2007.

[15] V.Raghavan, P.Bollmann, and G.S.Jung. A critical investigation of recall and precision as measures of retrieval system performance. *ACM Transactions on Information Systems*, Vol.7, pp.205-229, 1989.

[16] C.Tang, S.Dwarkadas, and Z.Xu. On scaling latent semantic indexing for large peer-to-peer systems. *In Proceedings of the 27th Annual International ACM SIGIR Conference*, Sheffield, UK, 2004

[17] K.Wang and J.Zhang. MSP:A class of parallel multistep successive sparse approximate inverse preconditioning strategies. *SIAM Journal on Scientific Computing*, Vol. 24, No.4, pp. 1141-1151, 2003

[18] C. Shen and D. Williams. Approximate Matrix Decomposition Technniques in Information Retrieval, In *Proceedings of World Congress on Engineering and Computer Science 2007*, pp. 347 - 352, October 24-26, 2007.