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 sparse 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 [5] 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][8][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...
of the concept matrix $C$. We write the concept decomposition as an $m \times n$ matrix

$$\tilde{A} = CM,$$

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

$$M = \arg \min_M \| A - CM \|_F^2.$$

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.,

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

### 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

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

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 $f(M) = \| A - CM \|_F^2$ 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 $\tilde{M}_i$, $i = 1, 2, \ldots$ 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
4.2 Multistep approach

Yields an approximate decomposition matrix \( n \) has many rows that are identically zero \([4, 17]\). After removing the squares problem (8) is solved by first computing an in which \( m \) small rectangular matrix. It has full rank if the matrix \( C \) corresponding to \( j \) can be computed for each \( j \), we have a reduced matrix \( \bar{C} ) \) 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 \( m_j \) and the \( n_2 \) columns of \( C \) corresponding to \( m_j \) 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 \( n_1 \times n_2 \)

\[
\bar{C}_j \tilde{m}_j - \bar{a}_j \|_2, \quad j = 1, 2, \ldots, n,
\]

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 \([9]\). 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

\[
\bar{C}_j = Q_j \begin{pmatrix} R_j^1 \\ 0 \end{pmatrix},
\]

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 \([9]\) is solved by first computing \( \bar{c}_j = Q_j^T \bar{a}_j \) and then obtaining the solution as \( \tilde{m}_j = R_j^{-1} \bar{c}_j (1 : n_2) \). In this way, \( m_j \) 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

\[
CM = A
\]

by using our modified sparse approximate inverse techniques developed in \([13]\). If \( CM_1 \) is not close enough to \( A \), we can compute another approximate sparse matrix \( M_2 \) for the equation

\[
CM = A - CM_1
\]

Based on Eq. (11), \( CM_1 \approx A - CM_1 \), 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

\[
CM = A - C(M_1 + M_2)
\]

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 \( M_1, M_2, \ldots, M_k \) such that \( C(M_1 + M_2 + \ldots + M_k) \approx A \). As \([17]\) pointed out, a few sparse matrices \( M_{k-1}, M_{k-2}, \ldots, M_1 \) 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

\[
\lim_{l \to \infty} C(M_1 + M_2 + \ldots + M_l) = A
\]

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 term values of both \( C \) and \( A \) is described below.
We determined the sparsity pattern of $M_1$ column by column. First, find the largest entry in each column of $A$, suppose they are $a_{11}$, $a_{12}$, $a_{13}$, $a_{14}$, and $a_{15}$ in Eq. (1). Then the sparsity pattern of $m_1^{(1)}$, is the same as that of $c_1$ 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:

$$
\begin{pmatrix}
  x & 0 & 0 & x & 0 \\
  0 & x & 0 & x & 0 \\
  x & 0 & x & 0 & 0 \\
  0 & x & 0 & x & 0 
\end{pmatrix}
$$

(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 $a_{11}$, $a_{12}$, $a_{13}$, $a_{14}$, and $a_{15}$ in Eq. (1). Then the sparsity pattern of $m_1^{(1)}$, is the same as that of $c_1$ 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:

$$
\begin{pmatrix}
  x & 0 & 0 & x & 0 \\
  0 & x & 0 & x & 0 \\
  x & 0 & x & 0 & 0 \\
  0 & x & 0 & x & 0 
\end{pmatrix}
$$

(13)

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 $M$ 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 $\|A - CM_1\|_2$ 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. 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%, ..., 90%. Then we compare their storage costs by listing the number of nonzeros of the approximate matrix $M$ obtained in Eq. (4). 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 $M$ 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".

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 $nstep = 2,3$ 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 $k = 500$.
In terms of query precisions, query time and memory costs.

Number of queries $n_{\mathrm{step}}$ 1 30 - 2 4 2 86,761 17 11 2 112 2 4 approach. The data listed in the last two rows of Table 3 are 

Again, the higher values of $M$ lead to more work, more lead to better performance. Besides, the threshold value of $n_{\mathrm{e}}$ 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. 6 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, \( k = 500 \)

<table>
<thead>
<tr>
<th>Methods</th>
<th>ne</th>
<th>nstep</th>
<th>nnz</th>
<th>recall 10%</th>
<th>40%</th>
<th>80%</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSP</td>
<td>2</td>
<td>1</td>
<td>49,974</td>
<td>0.3377</td>
<td>0.1897</td>
<td>0.0603</td>
</tr>
<tr>
<td>MSSP</td>
<td>2</td>
<td>2</td>
<td>91,044</td>
<td>0.3629</td>
<td>0.2181</td>
<td>0.0702</td>
</tr>
<tr>
<td>SSP_4</td>
<td>4</td>
<td>1</td>
<td>107,480</td>
<td>0.3580</td>
<td>0.2007</td>
<td>0.08649</td>
</tr>
<tr>
<td>MSSP</td>
<td>2</td>
<td>3</td>
<td>124,559</td>
<td>0.3831</td>
<td>0.2202</td>
<td>0.0724</td>
</tr>
<tr>
<td>CPM</td>
<td>--</td>
<td>--</td>
<td>699,000</td>
<td>0.4086</td>
<td>0.2390</td>
<td>0.0759</td>
</tr>
</tbody>
</table>

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 [13]. 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 \( f(M) = \min \| A - CM \|^2_F \).

7. REFERENCES


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.


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 $k = 800$.


