Two Uses for Updating the Partial Singular Value Decomposition in Latent Semantic Indexing

Jane E. Tougas [1]

Faculty of Computer Science, Dalhousie University, Halifax, NS, B3H 1W5, Canada

Raymond J. Spiteri [2]

Department of Computer Science, University of Saskatchewan, Saskatoon, SK, S7N 5C9, Canada

Abstract

Latent Semantic Indexing (LSI) is an information retrieval (IR) method that connects IR with numerical linear algebra by representing a dataset as a *term-document matrix.* Because of the tremendous size of modern databases, such matrices can be very large. The partial singular value decomposition (PSVD) is a matrix factorization that captures the salient features of a matrix, while using much less storage. We look at two challenges posed by this PSVD data compression process in LSI. Traditional methods of computing the PSVD are very expensive; most of the processing time in LSI is spent in calculating the PSVD of the term-document matrix. Thus, the first challenge is calculating the PSVD efficiently, in terms of computational and memory requirements. The second challenge is efficiently updating the PSVD when the matrix is altered slightly. In a rapidly expanding environment, such as the Internet, the term-document matrix is altered often as new documents and terms are added. Updating the PSVD of this matrix is much more efficient than recalculating it after each change. We investigate the use of the PSVD updating methods proposed by Zha and Simon (1999, SIAM J. Sci. Comput., 21, 2) to meet both of these challenges. Results are presented illustrating that updating in this manner provides tremendous savings in computation time, with little or no significant reduction in accuracy. An algorithm for iteratively computing the PSVD of a matrix using the document updating method will then be presented. This iterative method, suggested by Zha and Zhang (1999, SIAM J. Matrix Anal. Appl., 21, 2), provides a means of calculating the PSVD for matrices so large that the computation would be infeasible using traditional methods. Again, results are given showing that this method provides savings in computational time and memory resources without compromising the accuracy of the results.

Key words: partial singular value decomposition, updating, latent semantic indexing, iterative methods
1991 MSC: 15A18

1 Introduction

With the tremendous increase in the size of both the Internet and modern databases comes a corresponding increase in the need for efficient information retrieval (IR) methods. Latent Semantic Indexing (LSI) [7] is an IR method that relies heavily on techniques from numerical linear algebra. LSI uses a mathematical approach known as the *vector-space model*. The vector-space model represents a document collection as a *term-document matrix* containing a column vector for each document in the text collection and a row vector for each semantically significant term [10]. Terms that occur in most of the documents in a text collection are not considered semantically significant because they are not useful in differentiating between documents. Typically, words that occur in at least 80% of the documents are considered to be semantically insignificant *stopwords*; consequently they are not included in the term-document matrix [1]. Common stopwords include words such as *and*, *a*, and *the*. A list of 571 such English stopwords is used by the SMART retrieval system at Cornell University [6].

A term-document matrix $\mathbf{A}$ thus has t rows and d columns, where t is the number of semantically significant terms, and d is the number of documents. Each entry a_{ij} indicates the importance of term i relative to document j, where $1 \leq i \leq t$, and $1 \leq j \leq d$. The entries may simply be binary (0 if the term does not appear in the document and 1 otherwise), raw term

Email addresses: `tougas@cs.dal.ca` (Jane E. Tougas), `spiteri@cs.usask.ca` (Raymond J. Spiteri).
[1] The work of this author is supported by NSERC Canada and the Killam Foundation.
[2] The work of this author is supported by a grant from NSERC Canada.

frequencies (the number of times the term appears in the document), or more typically, weighted term frequencies [1], [9]. A user's query is represented as a document (column) vector using the same stopword removal and weighting scheme that have been applied to the document collection. The vectors of documents (and queries) with many terms in common will be close together in the t-dimensional vector space; conversely the vectors of documents with few terms in common will be far apart. This distance, or *similarity*, is typically measured using the cosine of the angle between each pair of vectors; this is known as the *cosine similarity measure* [1], [2]. The cosine of the angle between two vectors that are very close together will be large (close to 1), whereas the cosine of the angle between two vectors that are far apart will be small.

Unfortunately, the retrieval of text is complicated by the fact that many words have more than one meaning (they are *polysemous*). When a polysemous word is used in a search query, irrelevant documents about the word's other meaning(s) may be retrieved. This is known as *precision failure.* A further complication arises from the fact that many words have similar meanings (they are *synonymous*). When a word that has a synonym is used in a search query, relevant documents containing a synonym, but not the specific word used in the query, may be overlooked. This is known as *recall failure.* LSI uses a matrix factorization method known as the *partial singular value decomposition* (PSVD) to reduce the problems of recall and precision failure. Using the PSVD, the data in the term-document matrix are projected into a lower-dimensional vector space. This has the effect of removing noise (caused by factors such as synonymy and polysemy) from the data and giving a better representation of the text collection. Query vectors are also projected into the lower-dimensional space using the PSVD. Although research indicates that LSI is more successful in dealing with the problems caused by synonymy than those caused by polysemy [7], this does not detract from the importance of LSI in IR. LSI retrieves the documents that are most similar to a search query (those closest to the query in the vector space), even if the documents do not contain all (or any) of the terms contained in the query.

Although using the PSVD to project the term-document matrix into a lower-dimensional space has the benefit of removing noise from the data, it has the drawback of being computationally expensive. In fact, even using the most advanced numerical linear algebra techniques, the majority of the processing time in LSI is taken up with computing the PSVD [3], [4]. With a dynamic medium such as the Internet, the term-document matrix undergoes frequent changes as new documents and terms are added. Given the potentially huge size of such term-document matrices, recomputing the PSVD each time the term-document matrix is altered can be prohibitively expensive. Traditionally, a method known as *folding-in* has been used to modify the PSVD when recomputing it is too costly. Unfortunately, although the folding-in method is much faster than recomputing the PSVD, it may result in a significant degradation of retrieval performance [4], [13]. A more recent and more accurate approach is to *update* the PSVD using updating algorithms introduced by Zha and Simon [13]. Updating the PSVD is a compromise between recomputing the PSVD and folding-in: although the updating method is slower than the folding-in method, it is much faster than recomputing the PSVD, and yet it does not significantly degrade retrieval performance the way the folding-in method may. Although updating methods have also been proposed by O'Brien [8] and Berry, Dumais, and O'Brien [4], Zha and Simon have shown that these methods give inferior results when compared to the methods they introduce in [13]. In the absence of roundoff errors, the updating procedure of [13] produces the *exact* PSVD of the updated matrix. In practice roundoff errors do affect the results. However, these did not result in a significant change in the performance of LSI in a number of experiments that we have carried out [11].

The purpose of this paper is not only to show that updating the PSVD has definite advantages over recomputing the PSVD or using the folding-in method, but also to show that the updating method may be used to compute the PSVD of a matrix [14]. This is an especially important technique for the case in which the matrix is so large that memory constraints may prohibit the use of traditional PSVD methods.

The remainder of the paper proceeds as follows. Section 2 presents background material on the PSVD and the folding-in method, Section 3 describes the algorithms for the updating method, and Section 4 gives experimental results comparing the methods of recomputing the PSVD, folding-in, and updating the PSVD. Section 5 discusses the use of the updating method to compute the PSVD, Section 6 presents results of experiments using the updating method to compute the PSVD, and Section 7 presents our conclusions.

2 Background

2.1 SVD and PSVD

In order to understand the PSVD, it is helpful to first understand the *singular value decomposition* (SVD). The SVD is a matrix factorization that in various senses captures the most important characteristics of a matrix. Given a matrix $\mathbf{A}$ with t rows and d columns, its SVD has the form $\mathbf{A} = \mathbf{U\Sigma V^T}$, where $\mathbf{U}$ is an orthogonal matrix with t rows and columns, and $\mathbf{V}$ is an orthogonal matrix with d rows and columns. $\mathbf{U}$ and $\mathbf{V}$ contain the left and right *singular vectors* of $\mathbf{A}$ respectively. When $\mathbf{A}$ is a term-document matrix, the left singular vectors represent the term vectors, and the right singular vectors represent the document vectors. The matrix $\mathbf{\Sigma}$ has non-zero entries only on the diagonal, although not all diagonal entries are necessarily non-zero. These diagonal entries are the *singular values* of $\mathbf{A}$. The singular values are in non-increasing order, and are denoted σ_j, for $j = 1, 2, \ldots, \min(t, d)$. The number of non-zero singular values is the rank, r of the matrix. See Figure 1 for a schematic depiction of the SVD.

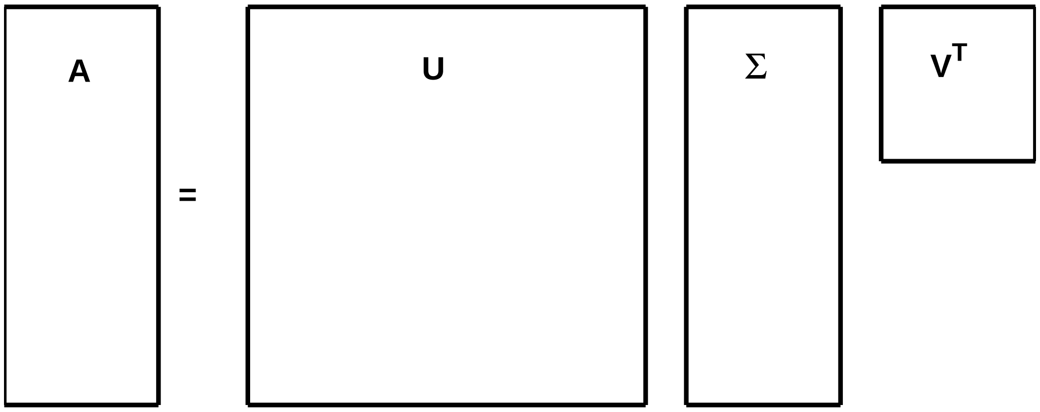

Fig. 1. The SVD of $\mathbf{A}$

An alternative way to represent the SVD of a matrix is as the sum of r rank-one matrices $\mathbf{A} = \sum_{j=1}^{r} \sigma_j \mathbf{u}_j \mathbf{v}_j^T$, where $\mathbf{u}_j$ and $\mathbf{v}_j$ are the jth columns of matrices $\mathbf{U}$ and $\mathbf{V}$, respectively. This representation of the SVD allows the formation of optimal lower-rank approximations of $\mathbf{A}$. These lower-rank approximations are optimal in the sense that for a given rank k, where $0 \leq k < r$, there is no matrix of rank at most k that is closer to $\mathbf{A}$ as measured in the 2-norm or the Frobenius norm; see, e.g., [12]. Let matrices $\mathbf{U}_k$ and $\mathbf{V}_k$ be the first k columns of $\mathbf{U}$ and $\mathbf{V}$ respectively, and let matrix $\mathbf{\Sigma}_k$ be the leading submatrix of $\mathbf{\Sigma}$ with k rows and k columns. Then $\mathbf{A}_k = \mathbf{U}_k \mathbf{\Sigma}_k \mathbf{V}_k^T$ is the optimal approximation (of rank at most k) of $\mathbf{A}$. This is the partial SVD (PSVD) of $\mathbf{A}$; see Figure 2. This approximation can be used to reduce the dimension of the term-document matrix, while eliciting the underlying structure of the data. In LSI, the effect of this dimensional reduction on the data is a muting of the noise caused by synonymy and an enhancing of the latent patterns that indicate semantically similar terms. This means that $\mathbf{A}_k$ can actually be a better representation of the data than the original term-document matrix. Note that in LSI, it is not necessary to explicitly form $\mathbf{A}_k$; the matrices $\mathbf{U}_k$, $\mathbf{\Sigma}_k$, and $\mathbf{V}_k$ are used instead. The optimal number of dimensions k (singular values and corresponding singular vectors) to keep in the reduced term-document matrix varies, but experiments indicate that between 100 and 300 give the best results [4]. This tremendous dimensional reduction demonstrates the power of the PSVD as a method of data compression.

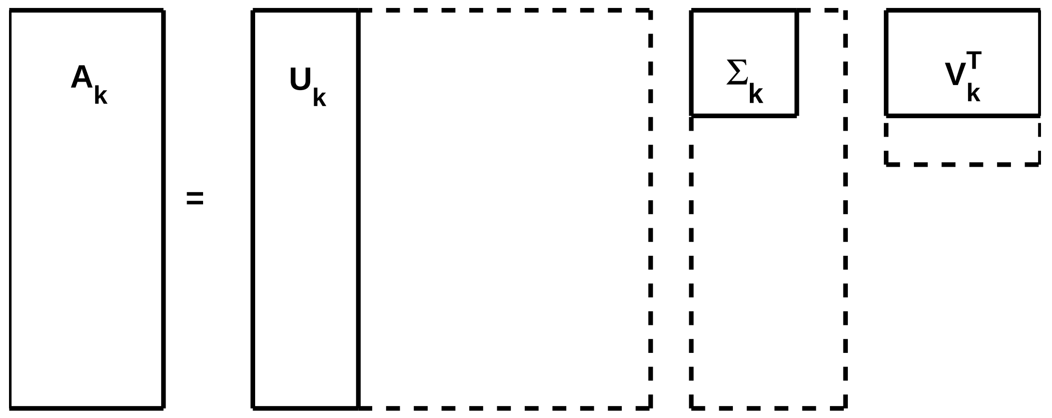

Fig. 2. The PSVD of $\mathbf{A}$

2.2 Recomputing and Folding-in

Let $\mathbf{D} \in \Re^{t\times p}$ contain p document vectors to be appended to the term-document matrix $\mathbf{A}$, and let $\mathbf{A}_k = \mathbf{U}_k\mathbf{\Sigma}_k\mathbf{V}_k^T$ be the PSVD of $\mathbf{A}$. The recomputing method simply recalculates the PSVD of the term-document matrix from scratch each time changes are made to the matrix. Although this is very accurate, it is also very expensive.

The folding-in method is computationally inexpensive compared to recomputing the PSVD. The folding-in method projects new documents into the lower-dimensional vector space by performing a matrix multiplication, $\mathbf{D}_k = \mathbf{D}^T\mathbf{U}_k\mathbf{\Sigma}_k^{-1}$, and then appending the result, $\mathbf{D}_k \in \Re^{t\times p}$, to the bottom of matrix $\mathbf{V}_k$. This gives the modified matrix $\hat{\mathbf{V}}_k \in \Re^{(d+p)\times k}$. Note that matrices $\mathbf{U}_k$ and $\mathbf{\Sigma}_k$ are not changed with this method. This means that as more and more documents are folded in, the representation of the dataset becomes less and less accurate.

The folding-in method projects new terms into a lower-dimensional vector space in a similar fashion. Let $\mathbf{T} \in \Re^{q\times d}$ contain q term (row) vectors to be appended to the term-document matrix $\mathbf{A}$. The folding-in method projects the new terms into the lower-dimensional vector space by performing a matrix multiplication $\mathbf{T}_k = \mathbf{T}\mathbf{V}_k\mathbf{\Sigma}_k^{-1}$, and then appending the result, $\mathbf{T}_k \in \Re^{q\times k}$, to the bottom of $\mathbf{U}_k$. This gives the modified matrix $\hat{\mathbf{U}}_k \in \Re^{(t+p)\times k}$. As with the folding-in of documents, matrices $\mathbf{V}_k$ and $\mathbf{\Sigma}_k$ are not changed.

3 Updating the PSVD

Zha and Simon's method for updating the PSVD [13] of a term-document matrix is more complicated and more computationally expensive than the folding-in method. However, in the absence of round-off error, this updating gives the exact PSVD of the modified term-document matrix [13]. Moreover, it is much less computationally intense than recomputing the PSVD anew. Although each update requires both a QR-factorization and an SVD calculation, both of these computations are performed on relatively small intermediate matrices (determined by the size of k). These calculations are thus relatively inexpensive, especially compared to recomputing the PSVD of the entire matrix.

A typical scenario for using the updating methods in LSI is that first, new documents are added to the document collection, and the PSVD of the term-document matrix is updated to reflect the addition of these documents. If the addition of these documents increases the number of terms, the PSVD of the term-document matrix is then updated to reflect the addition of the new terms. If the addition of these new documents and terms affects the weights in the term-document matrix, then as a final step, the PSVD of the term-document matrix is updated to reflect the corresponding changes to the term-weights. Sections 3.1–3.3 detail each of these updating algorithms [13] in turn. In each case, let $\mathbf{I}_n$ represent the identity matrix of dimension $n \times n$, and let $\mathbf{A}_k = \mathbf{U}_k \mathbf{\Sigma}_k \mathbf{V}_k$ be the rank-k PSVD of the term-document matrix $\mathbf{A}$ of dimension $t \times d$.

3.1 Updating when Documents are added to the Term-document Matrix

Assume that there are p documents to be appended to an existing term-document matrix $\mathbf{A} \in \Re^{t \times d}$. Let $\mathbf{D} \in \Re^{t \times p}$ be the term-document matrix containing these document vectors. The following method updates the PSVD of $\mathbf{A}$ to give the PSVD of $\tilde{\mathbf{A}}$, where $\tilde{\mathbf{A}} = [\mathbf{A}, \mathbf{D}]$ is the updated term-document matrix.

Let $\hat{\mathbf{D}} \in \Re^{t \times p}$ be defined by $\hat{\mathbf{D}} = \left(\mathbf{I}_t - \mathbf{U}_k \mathbf{U}_k^T\right) \mathbf{D}$, and then form the (reduced) QR decomposition of $\hat{\mathbf{D}}$ such that $\mathbf{Q}_\mathbf{D} \mathbf{R}_\mathbf{D} = \hat{\mathbf{D}}$. Recall that with such a decomposition, $\mathbf{Q}_\mathbf{D} \in \Re^{t \times p}$ has orthogonal columns, and $\mathbf{R}_\mathbf{D} \in \Re^{p \times p}$ is upper triangular. Then,

$$\tilde{\mathbf{A}} = [\mathbf{A}, \mathbf{D}] \approx [\mathbf{A}_k, \mathbf{D}] = [\mathbf{U}_k, \mathbf{Q}_\mathbf{D}] \begin{bmatrix} \mathbf{\Sigma}_k & \mathbf{U}_k^T \mathbf{D} \\ \mathbf{0} & \mathbf{R}_\mathbf{D} \end{bmatrix} \begin{bmatrix} \mathbf{V}_k^T & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_p \end{bmatrix}.$$

Let $\hat{\mathbf{A}} \in \Re^{(k+p)\times(k+p)}$ be defined by $\hat{\mathbf{A}} = \begin{bmatrix} \mathbf{\Sigma}_k & \mathbf{U}_k^T\mathbf{D} \\ \mathbf{0} & \mathbf{R}_\mathbf{D} \end{bmatrix}$.

Now form the SVD of $\hat{\mathbf{A}}$, and partition it to give

$$\hat{\mathbf{A}} = \left[\hat{\mathbf{U}}_k, \hat{\mathbf{U}}_p\right] \begin{bmatrix} \hat{\mathbf{\Sigma}}_k & \mathbf{0} \\ \mathbf{0} & \hat{\mathbf{\Sigma}}_p \end{bmatrix} \left[\hat{\mathbf{V}}_k, \hat{\mathbf{V}}_p\right]^T,$$

where $\hat{\mathbf{U}}_k \in \Re^{(k+p)\times k}$, $\hat{\mathbf{U}}_p \in \Re^{(k+p)\times p}$, $\hat{\mathbf{\Sigma}}_k \in \Re^{k\times k}$, $\hat{\mathbf{\Sigma}}_p \in \Re^{p\times p}$, $\hat{\mathbf{V}}_k \in \Re^{(k+p)\times k}$, and $\hat{\mathbf{V}}_p \in \Re^{(k+p)\times p}$.

Then the rank-k PSVD of the updated term-document matrix $\tilde{\mathbf{A}} = [\mathbf{A}, \mathbf{D}]$ is

$$\tilde{\mathbf{A}}_k = \left([\mathbf{U}_k, \mathbf{Q}_\mathbf{D}]\,\hat{\mathbf{U}}_k\right)\hat{\mathbf{\Sigma}}_k \left(\begin{bmatrix} \mathbf{V}_k & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_p \end{bmatrix}\hat{\mathbf{V}}_k\right)^T.$$

3.2 Updating when Terms are added to the Term-document Matrix

Assume that there are q terms to be appended to an existing term-document matrix $\mathbf{A} \in \Re^{t\times d}$. Let $\mathbf{T} \in \Re^{q\times d}$ be the term-document matrix containing these term (row) vectors. The following method updates the PSVD of $\mathbf{A}$ to give the PSVD of $\tilde{\mathbf{A}}$, where

$$\tilde{\mathbf{A}} = \begin{bmatrix} \mathbf{A} \\ \mathbf{T} \end{bmatrix} \qquad \text{is the updated term-document matrix.}$$

Let $\hat{\mathbf{T}} \in \Re^{d\times q}$ be defined by $\hat{\mathbf{T}} = \left(\mathbf{I}_d - \mathbf{V}_k\mathbf{V}_k^T\right)\mathbf{T}^T$, and then form the (reduced) QR decomposition of $\hat{\mathbf{T}}$ such that $\mathbf{Q}_\mathbf{T}\mathbf{R}_\mathbf{T} = \hat{\mathbf{T}}$. Then $\mathbf{Q}_\mathbf{T} \in \Re^{d\times q}$ has orthogonal columns, $\mathbf{R}_\mathbf{T} \in \Re^{q\times q}$ is upper triangular, and

$$\tilde{\mathbf{A}} = \begin{bmatrix} \mathbf{A} \\ \mathbf{T} \end{bmatrix} \approx \begin{bmatrix} \mathbf{A}_k \\ \mathbf{T} \end{bmatrix} = \begin{bmatrix} \mathbf{U}_k & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_q \end{bmatrix} \begin{bmatrix} \mathbf{\Sigma}_k & \mathbf{0} \\ \mathbf{T}\mathbf{V}_k & \mathbf{R}_\mathbf{T}^T \end{bmatrix} [\mathbf{V}_k, \mathbf{Q}_\mathbf{T}]^T.$$

Let $\quad \hat{\mathbf{A}} \in \Re^{(k+q)\times(k+q)}$ be defined by $\hat{\mathbf{A}} = \hat{\mathbf{A}} = \begin{bmatrix} \mathbf{\Sigma}_k & \mathbf{0} \\ \mathbf{T}\mathbf{V}_k & \mathbf{R}_{\mathbf{T}}^T \end{bmatrix}.$

Now form the SVD of $\hat{\mathbf{A}}$, and partition it to give

$$\hat{\mathbf{A}} = \left[\bar{\mathbf{U}}_k, \bar{\mathbf{U}}_q\right] \begin{bmatrix} \bar{\mathbf{\Sigma}}_k & \mathbf{0} \\ \mathbf{0} & \bar{\mathbf{\Sigma}}_q \end{bmatrix} \left[\bar{\mathbf{V}}_k, \bar{\mathbf{V}}_q\right]^T,$$

where $\bar{\mathbf{U}}_k \in \Re^{(k+q)\times k}$, $\bar{\mathbf{U}}_q \in \Re^{(k+q)\times q}$, $\bar{\mathbf{\Sigma}}_k \in \Re^{k\times k}$, $\bar{\mathbf{\Sigma}}_q \in \Re^{q\times q}$, $\bar{\mathbf{V}}_k \in \Re^{(k+q)\times k}$, and $\bar{\mathbf{V}}_q \in \Re^{(k+q)\times q}$.

Then the rank-k PSVD of the updated term-document matrix

$$\tilde{\mathbf{A}} = \begin{bmatrix} \mathbf{A} \\ \mathbf{T} \end{bmatrix} \qquad \text{is} \qquad \tilde{\mathbf{A}}_k = \left(\begin{bmatrix} \mathbf{U}_k & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_q \end{bmatrix} \bar{\mathbf{U}}_k\right) \bar{\mathbf{\Sigma}}_k \left(\left[\mathbf{V}_k, \mathbf{Q}_{\mathbf{T}}\right] \bar{\mathbf{V}}_k\right)^T.$$

3.3 Updating Term-weights in the Term-document Matrix

Assume that there are s terms whose weights need to be adjusted in an existing term-document matrix $\mathbf{A} \in \Re^{t\times d}$. Let $\mathbf{S} \in \Re^{t\times s}$ be the *selection* matrix. A selection matrix has a column for each term whose weight must be modified: each column has one entry which is 1 (to select the term), and all other entries 0. For example if term i is represented by column j, then the entry s_{ij} will be 1, and all other entries in column j will be 0. Let $\mathbf{W} \in \Re^{d\times s}$ be the matrix such that each column $\mathbf{w}_i$ contains the difference between the old term weights and the new term weights for the term i. The following method updates the PSVD of $\mathbf{A}$ to give the PSVD of $\tilde{\mathbf{A}}$, where $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{S}\mathbf{W}^T$ is the adjusted term-document matrix.

To begin, let $\hat{\mathbf{S}} \in \Re^{t\times s}$ be defined by $\hat{\mathbf{S}} = \left(\mathbf{I}_t - \mathbf{U}_k\mathbf{U}_k^T\right)\mathbf{S}$, and let $\hat{\mathbf{W}} \in \Re^{d\times s}$ be defined by $\hat{\mathbf{W}} = \left(\mathbf{I}_d - \mathbf{V}_k\mathbf{V}_k^T\right)\mathbf{W}$. Now form the (reduced) QR decomposition of $\hat{\mathbf{S}}$ such that $\mathbf{Q}_{\mathbf{S}}\mathbf{R}_{\mathbf{S}} = \hat{\mathbf{S}}$. Then, $\mathbf{Q}_{\mathbf{S}} \in \Re^{t\times s}$ has orthogonal columns, and $\mathbf{R}_{\mathbf{S}} \in \Re^{s\times s}$ is upper triangular. Also form the (reduced) QR decomposition of $\hat{\mathbf{W}}$ such that $\mathbf{Q}_{\mathbf{W}}\mathbf{R}_{\mathbf{W}} = \hat{\mathbf{W}}$. Then $\mathbf{Q}_{\mathbf{W}} \in \Re^{d\times s}$ has orthogonal columns, and $\mathbf{R}_{\mathbf{W}} \in \Re^{s\times s}$ is upper triangular. Using these decompositions,

$$\begin{aligned}\tilde{\mathbf{A}} &= \mathbf{A} + \mathbf{S}\mathbf{W}^T \\ &\approx \mathbf{A}_k + \mathbf{S}\mathbf{W}^T = [\mathbf{U}_k, \mathbf{Q}_{\mathbf{S}}] \left(\begin{bmatrix} \mathbf{\Sigma}_k & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} + \begin{bmatrix} \mathbf{U}_k^T\mathbf{S} \\ \mathbf{R}_{\mathbf{S}} \end{bmatrix} \begin{bmatrix} \mathbf{V}_k^T\mathbf{W} \\ \mathbf{R}_{\mathbf{W}} \end{bmatrix}^T \right) [\mathbf{V}_k, \mathbf{Q}_{\mathbf{W}}]^T .\end{aligned}$$

Let $\hat{\mathbf{A}} \in \Re^{(k+s)\times(k+q)}$ be defined by $\hat{\mathbf{A}} = \begin{bmatrix} \mathbf{\Sigma}_k & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} + \begin{bmatrix} \mathbf{U}_k^T\mathbf{S} \\ \mathbf{R}_{\mathbf{S}} \end{bmatrix} \begin{bmatrix} \mathbf{V}_k^T\mathbf{W} \\ \mathbf{R}_{\mathbf{W}} \end{bmatrix}^T .$

Now form the SVD of $\hat{\mathbf{A}}$, and partition it to give

$$\hat{\mathbf{A}} = \left[\tilde{\mathbf{U}}_k, \tilde{\mathbf{U}}_s\right] \begin{bmatrix} \tilde{\mathbf{\Sigma}}_k & \mathbf{0} \\ \mathbf{0} & \tilde{\mathbf{\Sigma}}_s \end{bmatrix} \left[\tilde{\mathbf{V}}_k, \tilde{\mathbf{V}}_s\right]^T ,$$

where $\tilde{\mathbf{U}}_k \in \Re^{(k+s)\times k}$, $\tilde{\mathbf{U}}_s \in \Re^{(k+s)\times s}$, $\tilde{\mathbf{\Sigma}}_k \in \Re^{k\times k}$, $\tilde{\mathbf{\Sigma}}_s \in \Re^{s\times s}$, $\tilde{\mathbf{V}}_k \in \Re^{(k+s)\times k}$, and $\tilde{\mathbf{V}}_s \in \Re^{(k+s)\times s}$.

Then the rank-k PSVD of the updated term-document matrix $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{S}\mathbf{W}^T$ is

$$\tilde{\mathbf{A}}_k = \left([\mathbf{U}_k, \mathbf{Q}_{\mathbf{S}}]\, \tilde{\mathbf{U}}_k\right) \tilde{\mathbf{\Sigma}}_k \left([\mathbf{V}_k, \mathbf{Q}_{\mathbf{W}}]\, \tilde{\mathbf{V}}_k\right)^T .$$

4 Experiments: Updating the PSVD

In order to illustrate the accuracy of updating the PSVD in LSI, we compare the retrieval performance of an LSI implementation using this method with two alternative methods: one that recomputes the PSVD each time new documents are added, and one that folds-in new documents. In each case, we measure both the *average precision* and the CPU time. In information retrieval, *precision* is defined as the fraction of the retrieved documents that are relevant [1]. We average this precision over the number of queries, and then average again over the 11 *standard recall levels. Recall* is defined as the fraction of relevant documents that has been retrieved [1]; the standard recall levels are 0–10%, 10–20%, . . . , 90-100%. The maximum precision (averaged over the number of queries) in each recall level is used to compute the average over all the levels.

For these experiments, two document collections are used. The first is the MEDLINE text collection [5], containing 1033 documents and 30 queries. Removing semantically insignificant terms and stemming the remaining terms gives a term-document matrix $\mathbf{A}_{\text{MED}} \in \Re^{5735 \times 1033}$. The second text collection used is the CRANFIELD text collection [5], containing 1400 documents and 225 queries. For this collection, no stemming is done, but semantically insignificant words are removed, giving a term-document matrix $\mathbf{A}_{\text{CRAN}} \in \Re^{5321 \times 1400}$. For each text collection, we use a *term frequency inverse document frequency* (tfidf) weighting scheme [1] for both the document and query vectors. The similarity measure used in each of these experiments is the cosine of the angle between query and each of the document vectors.

All of the experiments in this section are run using *Matlab* Release 13 on an Ultra3 SunFire V880 (Solaris 8 operating system). In each experiment, we partition the term-document matrix for the whole text collection into an initial matrix and a number of smaller submatrices. The initial matrix is incrementally enlarged by iteratively appending the submatrices, and the average precision for each method is plotted at each increment. In each case, the PSVD of the initial matrix is computed using the `svds` function for sparse matrices in *Matlab*. For the MEDLINE text collection we choose $k = 125$, and for the CRANFIELD text collection we choose $k = 300$, where k is the number of dimensions (singular values and corresponding left and right singular vectors) computed. For the sake of brevity, the experiments described here use only document updating. We note that similar results are produced using term updating.

4.1 Experiments with the MEDLINE Text Collection

Figures 3–6 illustrate the results from experiments using the MEDLINE text collection. In each case the initial term-document matrix of 5735 terms and 433 documents has 600 documents added to it. Note that the initial matrix more than doubles in size as a result of the incremental additions. Figure 3 shows the average precision at each increment when there are 120 increments of 5 documents each, simulating a dynamic environment in which frequent small changes are made to the term-document matrix. As expected, Figure 3 indicates that the average precision for the folding-in method deteriorates rapidly compared with the other methods. The average precision for the updating method does not deteriorate until the initial matrix has approximately doubled in size, and even then the deterioration is very slight. In this example, although the updating method is computationally more expensive than the folding-in method, the results are essentially as good as recomputing. Moreover, the execution time is over 120 times less than recomputing. Table 1 contains the CPU times for these experiments.

Figure 4 shows the average precision at each increment for the case in which the initial term-document matrix with 433 documents has 600 documents added to it in 60 increments of 10 documents each. As in the previous experiment, Figure 4 shows that the average precision for the folding-in method deteriorates rapidly compared to the other methods. In this example the updating method gives similar results when compared to the method of recomputing the PSVD at each increment, but again using over 120 times less CPU time.

Figure 5 gives the average precision at each increment for the case in which 30 increments of 20 documents each are added to the initial matrix of 433 documents, and Figure 6 gives the average precisions for adding 15 increments of 40 documents each. As in the previous experiments, these figures show that the average precision for the folding-in method deteriorates rapidly compared to the other methods. In both cases the updating method gives similar results when compared to the method of recomputing at each increment, but the updating method requires over 100 times less CPU time than recomputing for Figures 5, and approximately 75 times less CPU time for Figure 6. See Table 1 for CPU times.

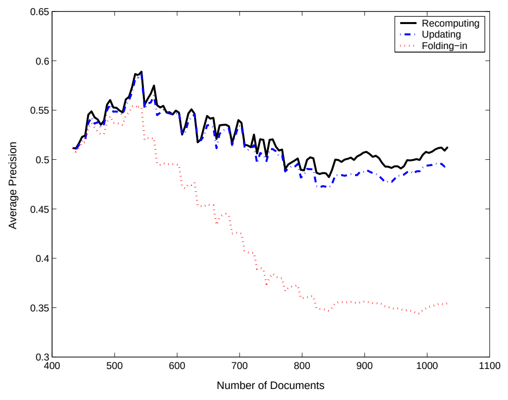

Fig. 3. Average precisions for three methods using MEDLINE collection: 600 documents added (120 groups of 5).

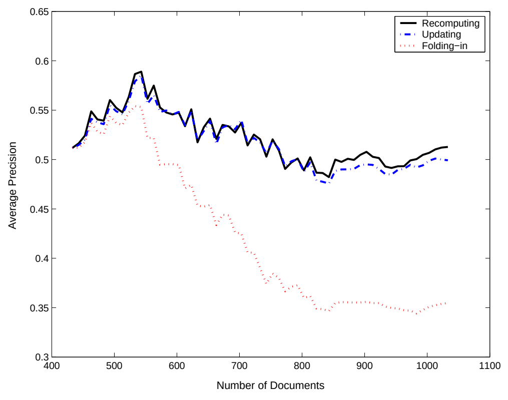

Fig. 4. Average precisions for three methods using MEDLINE collection: 600 documents added (60 groups of 10).

Method	CPU time	CPU time	CPU time	CPU time
	Increments of 5	Increments of 10	Increments of 20	Increments of 40
Recomputing	12689.14	6708.55	2879.01	1444.26
Updating	99.78	53.05	28.60	19.84
Folding-in	3.04	1.76	0.96	0.59

Table 1
MEDLINE CPU times (seconds): 600 documents added: groups of 5, 10, 20, 40.

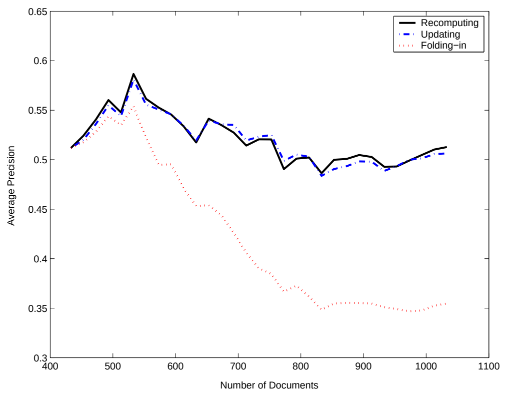

Fig. 5. Average precisions for three methods using MEDLINE collection: 600 documents added (30 groups of 20).

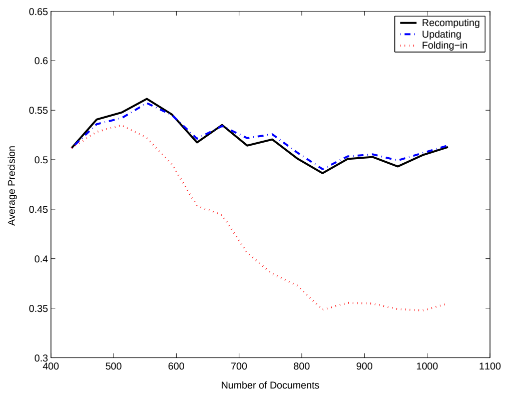

Fig. 6. Average precisions for three methods using MEDLINE collection: 600 documents added (15 groups of 40).

4.2 Experiments with the CRANFIELD Text Collection

Figures 7–10 illustrate experiments using the CRANFIELD text collection. In each case the initial term-document matrix of 5321 terms and 800 documents has 600 documents added to it. Figure 7 shows the average precision at each increment when there are 120 increments of 5 documents each. As expected, Figure 7 indicates that the average precision for the folding-in method deteriorates rapidly compared with the other methods. Although the updating method is computationally more expensive than the folding-in method, it gives much better results. In this example the updating method requires over 180 times less CPU time than recomputing; in this case recomputing takes almost 35 hours, whereas updating takes less than 12 minutes. Table 2 contains the CPU times.

Figure 8 shows the average precisions for the case in which the initial term-document matrix with 800 documents has 600 documents added to it in 60 increments of 10 documents each. Figure 8 shows that the average precision for the folding-in method deteriorates rapidly compared to the other methods. The updating method again gives much better results than the folding-in method. In this case the updating method takes less than 6 minutes, whereas recomputing takes more than 17 hours.

Figure 9 shows the average precisions for the case in which 30 increments of 20 documents each are added to the initial matrix of 800 documents, and Figure 10 gives the average precisions for adding 15 increments of 40 documents each. As in the previous experiments, these figures show that the average precision for the folding-in method deteriorates rapidly compared to the other methods. In both cases the updating method gives much better results than folding-in while being much faster than recomputing the PSVD at each increment. See Table 2 for CPU times.

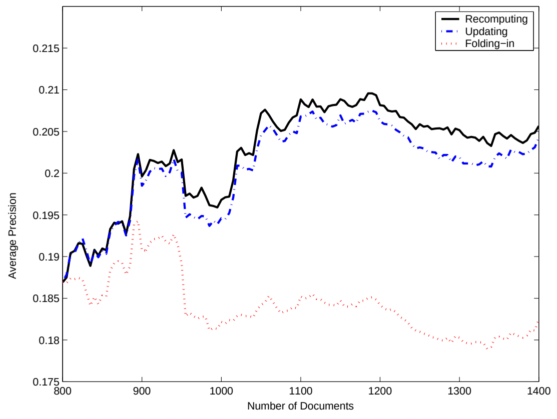

Fig. 7. Average precisions for three methods using CRANFIELD collection: 600 documents added (120 groups of 5).

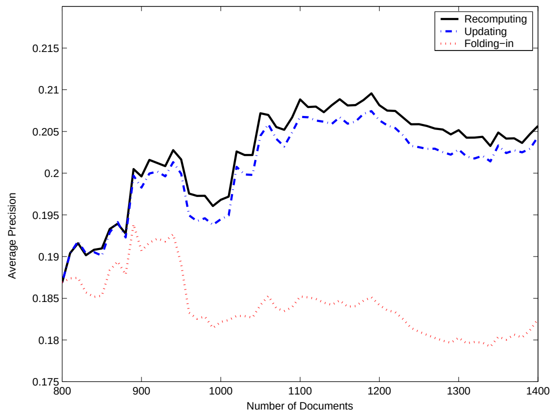

Fig. 8. Average precisions for three methods using CRANFIELD collection: 600 documents added (60 groups of 10).

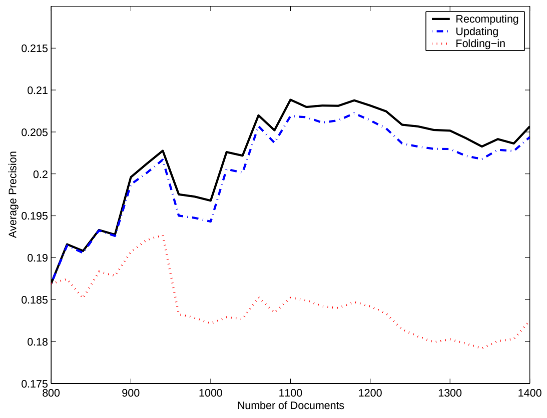

Fig. 9. Average precisions for three methods using CRANFIELD collection: 600 documents added (30 groups of 20).

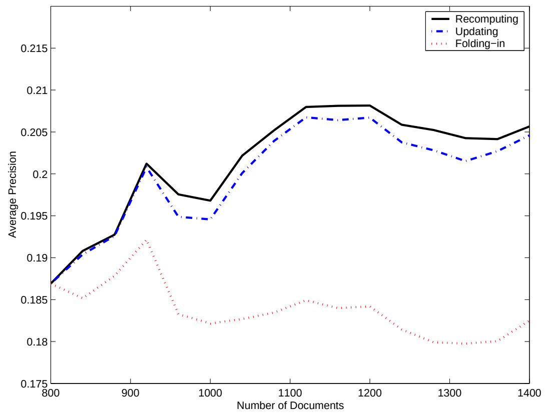

Fig. 10. Average precisions for three methods using CRANFIELD collection: 600 documents added (15 groups of 40).

Method	CPU time	CPU time	CPU time	CPU time
	Increments of 5	Increments of 10	Increments of 20	Increments of 40
Recomputing	125976.03	63530.89	31303.54	15781.51
Updating	689.80	353.29	216.81	102.00
Folding-in	15.28	8.50	4.49	2.65

Table 2
CRANFIELD CPU times (seconds): 600 documents added: groups of 5, 10, 20, 40.

5 Computing the PSVD using Updating Methods

Given the enormous size of both the Internet and many modern databases, it is common for a term-document matrix to be too large for the computation of its PSVD to be feasible with the available memory resources. In this case, using the updating method to compute the PSVD of the matrix is a viable alternative. The idea behind using the updating method to compute the PSVD of a matrix is very simple. The matrix is partitioned into manageable pieces (whose number and size depend on the available resources): $\mathbf{A} = [\mathbf{A}_1, \mathbf{A}_2, \cdots, \mathbf{A}_j]$. The PSVD of $\mathbf{A}_1$ is computed using a traditional method, such as the built-in `svds` function in *Matlab*. This PSVD is then updated $j-1$ times using the document updating method and submatrices $\mathbf{A}_2 \cdots \mathbf{A}_j$ in turn, to give an approximation to the PSVD of the original matrix $\mathbf{A}$. A strategy similar to term updating can be used for matrices that must be partitioned vertically rather than horizontally. Note that for the first submatrix $\mathbf{A}_1 \in \Re^{m \times n}$, it must be the case that $\min(m, n) > k$, where k is the number of singular values and singular vectors being computed. This is not a significant restriction in practice.

6 Experiments: Computing the PSVD using Updating Methods

In order to illustrate the accuracy of using the updating method to compute the PSVD in LSI, we compare the retrieval performance of an LSI implementation using this method with that of one which uses a traditional approach to computing the PSVD, in this case, the `svds` function in *Matlab*. In each case, we measure both the *average precision* and the CPU time. The same two document collections are used as in Section 4.

All of the experiments in this section are also run using *Matlab* Release 13 on an Ultra3 SunFire V880 (Solaris 8 operating system). In each experiment, we partition the term-document matrix for the whole text collection into an initial matrix and a number of additional submatrices. The initial matrix is incrementally enlarged by iteratively appending the submatrices, and the PSVD of each modified matrix is computed using the document updating method. In each case, the PSVD of the initial matrix is computed using the `svds` function for sparse matrices in *Matlab*. As before, for the MEDLINE text collection $k = 125$, and for the CRANFIELD text collection $k = 300$.

6.1 Experiments with the MEDLINE Text Collection

Table 3 gives the CPU times and average precisions for experiments with the MEDLINE text collection. The first row of the table gives the CPU time and average precision for the base case in which the document updating method is not used; i.e. the PSVD of the entire term-document matrix is computed using the *Matlab* function `svds`, and then the average precision using LSI is recorded. The experiments for the remaining rows use the document updating method to compute the PSVD of the term-document matrix. We see that as the number of submatrices increases (i.e., the number of updates performed increases) the CPU time for the process decreases; at some point (depending on the size of k) the overhead of the updating process causes the CPU times to increase as the number of partitions increases (see Section 6.2). The final row of the table shows a case in which the cost of the number of updates being performed outweighs the savings in performing updates on smaller submatrices, and thus the CPU time for performing 32 updates is slightly higher than that for performing only 16 updates. In other words, there is an optimal number of partitions somewhere between 16 and 32 updates. However, the *break-even point* at which updating becomes as expensive as computing the entire PSVD requires significantly more updates than are shown. Note that for these experiments, the average precision stays within 0.006, or 0.6% of that for the base case.

Number of Updates	CPU time (sec.)	Average Precision
0	107.68	51.28%
1	155.08	51.87%
2	134.21	51.83%
4	93.58	51.38%
8	49.72	51.07%
16	45.59	51.62%
32	57.57	51.85%

Table 3

6.2 Experiments with the CRANFIELD Text Collection

Table 4 gives the CPU times and average precisions for experiments with the CRANFIELD text collection. The results are qualitatively the same as for the MEDLINE text collection; there is an optimal number of partitions between 8 and 16, and the break-even point requires significantly more updates than are shown. Note that the average precision for all the examples is within .0019, or 0.19% of that of the base case in the first row.

Number of Updates	CPU time (sec.)	Average Precision
0	1132.70	20.57%
1	971.69	20.57%
2	741.90	20.62%
4	329.95	20.56%
8	316.21	20.40%
16	333.39	20.38%

Table 4

7 Conclusion

Latent Semantic Indexing (LSI) is an information retrieval (IR) method that represents a text collection as a term-document matrix and uses the PSVD of the matrix to project the data into a lower-dimensional space. Because of the tremendous size of modern databases, such a term-document matrix can potentially be very large. Traditional methods of computing the PSVD are very expensive, and in a rapidly expanding environment the term-document matrix is altered often as new documents and terms are added. We have demonstrated that the PSVD updating methods proposed by Zha and Simon [13] are effective in a dynamic environment in which there are many small updates made to the term-document matrix. This method of updating the PSVD achieves similar average precision to recomputing the PSVD, using only a fraction of the computation time. We have also demonstrated that the same updating methods may be used to compute the PSVD of a matrix by partitioning the matrix into submatrices, computing the PSVD of the first submatrix using a traditional method, and then using a PSVD updating method to iteratively update this PSVD to form the PSVD of the original matrix. This technique can offer savings in both memory resources and computation time, compared with using a traditional PSVD method, without compromising the accuracy of the results.

References

[1] R. A. Baeza-Yates, R. Baeza-Yates, and B. Ribeiro-Neto. *Modern Information Retrieval.* Addison-Wesley Longman Publishing Co., Inc., 1999.

[2] M. W. Berry and M. Browne. *Understanding Search Engines: Mathematical Modeling and Text Retrieval*. Software, Environments, and Tools. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, first edition, 1999.

[3] M. W. Berry, S. T. Dumais, and T. A. Letsche. Computational methods for intelligent information access, 1995. Presented at the Proceedings of Supercomputing.

[4] M. W. Berry, S. T. Dumais, and G. W. O'Brien. Using linear algebra for intelligent information retrieval. *SIAM Rev.*, 37(4):573–595, 1995.

[5] Cornell SMART System ftp://cs.cornell.edu/pub/smart.

[6] Cornell SMART System ftp://cs.cornell.edu/pub/smart/english.stop.

[7] S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman. Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, 41(6):391–407, 1990.

[8] G. W. O'Brien. Information tools for updating an SVD-encoded indexing scheme, 1994. Master's Thesis, The University of Knoxville, Tennessee.

[9] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Inf. Process. Manage.*, 24(5):513–523, 1988.

[10] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill Computer Science Series. McGraw-Hill, New York, NY, first edition, 1983.

[11] J. E. Tougas, H. Stern, and R. J. Spiteri. Updating the partial singular value decomposition in latent semantic indexing. Technical report, Department of Computer Science, University of Saskatchewan, 2005.

[12] L. N. Trefethen and D. Bau, III. *Numerical linear algebra*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1997.

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

[14] 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.