International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 09 Issue: 06 | Jun 2022 www.irjet.net p-ISSN: 2395-0072

Methodology for Managing Dynamic Collections on Semantic Semi-Structured XMLs

Hsu-Kuang Chang

I-Shou University No.1, Sec. 1, Syuecheng Rd. Dashu Township, Kaohsiung,Taiwan

---

Abstract - *Recently, the amount of XML document is in increasing as electronic document systems adopt XML as the standard format in document exchange. Like the weather, XML document databases rarely stay the same. Information is constantly added or removed, meaning that catalogs and indexes become obsolete or incomplete (sometimes in a matter of seconds). With great increase in online information, dynamic updating XML document takes a critical role in efficient document organization, navigation, and retrieval of a large amount of XML documents. XML document are modeled as a matrix, and a user's query and updating of the XML database is represented as a vector. Relevant XML documents in the XML database are then identified via vector operations. For the LSI model, latent semantic indexing, the most obvious approach to accommodating additions (new paths or documents) is to re-compute the SVD of the new path-by-document matrix, but, for large XML document databases, this procedure is very costly in time and space. Less expensive alternatives, folding-in combine with SVD-updating, have been presented in this paper. A new proposed method, folding-updating, is a combination of folding-in and updating the thin-SVD that is an even more attractive option. Folding-updating offers a significant improvement in computation time when compared with either re-computing the thin-SVD or updating the thin-SVD, and yet it results in little or no loss of accuracy.*

Key Words: thin-SVD, LSI, folding-in, SVD-updating, SVD-folding-updating, SVD-re-computing

1. INTRODUCTION

As the size of modern XML databases increases, the importance of having efficient methods of information retrieval (IR) increases accordingly. Latent Semantic Indexing (LSI) is an IR method that uses procedures from numerical linear algebra to represent a text collection as a term-document matrix [1]. The term-document matrix contains a column vector for each document in the text collection, and a row for each semantically significant term. LSI uses a matrix factorization method known as the thin singular value decomposition (thin-SVD). Unfortunately, traditional methods of computing the thin-SVD are computationally intensive; most of the processing time in LSI is spent in calculating the thin-SVD of the term-by-document matrix [2, 3]. In a dynamic environment, such as the Internet, the term-document matrix is altered often as new documents are added. Given the tremendous size of modern databases, re-computing the thin-SVD of the matrix each time such changes occur can be prohibitively expensive. LSI traditionally uses a method known as folding-in to modify the thin-SVD, in order to avoid re-computing the thin-SVD each time changes are made to the term-document matrix. The folding-in method has the benefit of being very fast, however its accuracy may degrade very quickly. A much more accurate approach is to update the thin-SVD using a method introduced by Zha and Simon [4]. This updating method modifies the thin-SVD of the term-document matrix to reflect the additions that are to be made to matrix. A new method, folding-updating, is a combination of folding-in and updating the thin-SVD that is an even more attractive option. Folding-updating offers a significant improvement in computation time when compared with either re-computing the thin-SVD or updating the thin-SVD, and yet it results in little or no loss of accuracy. We investigate the use of thin-SVD updating methods proposed by Zha and Simon [4]. Although updating methods have also been proposed by O'Brien [5] and Berry, Dumais and O'Brien [3], research indicates that these methods give inferior results when compared to the methods introduced by Zha and Simon [4]. The basic idea of LSI is that if two document vectors represent the same topic, they will share many associated words with a keyword and they will have very close semantic structures after dimension reduction via truncated SVD [10, 11]. Recent studies also indicate that retrieval accuracy of the truncated SVD technique can deteriorate if the document sets are large [12,13]. Several strategies have been proposed to deal with LSI on large datasets. The specification strategy was used to remove less important entries in truncated SVD matrices [14]. Clustered and distributed SVD strategies were proposed to partition large datasets [15]. Fan *et al.* (1999) [16] examined a random sampling based approach to SVD approximation and presented their results.

The rest of the paper is organized as follows. The section 2 gives a brief preprocessing XML documents into vector space model and overview of the SVD process. Section 3 outlines our incremental SVD algorithm. Section 4 presents our experimental procedure, results and discussion. The final section provides some concluding remarks and future research directions.

2. Preparation for Semantic-based XML Documents

2.1. Preprocessing XML documents

In this section, we first introduce pre-processing steps for the incorporation of hierarchical information in encoding the XML tree's paths. It is based on the preorder tree representation (PTR) [6] and will be introduced after a brief review of how to generate an XML tree from an XML document. To do so, we have to first go through the following five preprocessing steps for XML documents. The five preprocessing steps are conversion, path extraction, nested and duplicated path removal, similar element identification and transformation, and path elements encoding.

From five steps preprocess, now XML document is modeled as a XML tree $T=(V,E)$. T is connected tree with $V=\{ v_1, v_2, ...\}$ as a set of vertices and $v_1 \in V$, $v_2 \in V$, $(v_1, v_2) \in E$ as a set of edges. One distinguished vertex $r \in V$ is designated the root, and for all $v \in V$, there is a unique path from r to v. As an example, Figure 1 depicts a sample XML tree containing some information about collection of books. The *book* consists of *intro* tags, each comprising *title*, *author* and *date* tags. Each *author* contains *fname* and *lname*, each *date* includes *year* and *month* tags. Figure 1 left shows only the first letter of each tag for the simplicity.

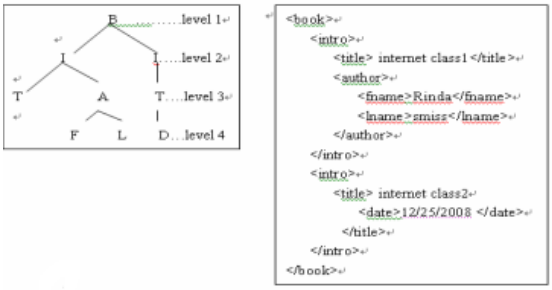


Figure 1 Example of XML Document

XML document has a hierarchical structure and this structure is organized with tag paths. Each tag path represents document characteristics that can predict the contents of XML document. Strictly speaking, it shows the semantic structural characteristics of XML document. In this paper, we propose a method for calculating the similarity using all tag paths of XML tree representing the semantic structural information of XML document. From now, a tag path as a term is called a ***path element***. Figure 3 shows path elements obtained from XML document of Figure 2.

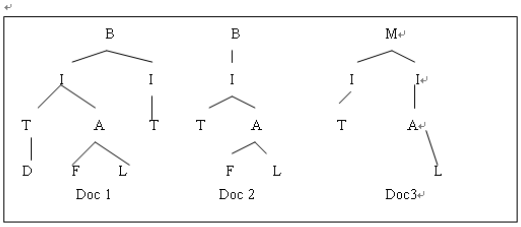


Figure 2 XML Documents Example

PE_{L-1}	PE_{L-2}	PE_{L-3}	PE_{L-4}
/B/I/T/D	/I/T/D	/T/D	D
/B/I/A/F	/I/A/F	/A/F	F
/B/I/A/L	/I/A/L	/A/L	L
/M/I/A/L	/I/T	/T	
/B/I/T/	/I/A	/A	
/B/I/A	/I		
/B/I/			
/B			
/M/I/T			
/M			
/M/I			

Figure 3 Path elements example

2.2. Modeling Document into Path Element Vector Space Model (PEVSM)

Vector model represents a document as a vector whose path elements are the weights of the elements within a document. In calculating the weight of each element within a document, Term Frequency and IDF (Inverse Document Frequency) method is used [7]. In this paper, we use path elements of XML tree as a term. And we propose the method to calculate the weights of path elements. We define PESSW (Path Element Structural Semantic Weight) that calculate the weight of a path element in a XML document. The PESSW is PEWF (Path Element Weighted Frequency) multiplied by PEIDF (Path Element Inverse Document Frequency). $PESSW_{ij}$ of ith path element in the jth document is shown in equation (1).

$$PESSW_{ij} = PEWF_{ij} \times PEIDF_{ij} \quad (1)$$

$PEWF_{ij}$ is shown in equation(2).

$$PEWF_{ij} = freq_{ij} \times \frac{1}{x^n} \qquad (2)$$

where, $freq_{ij}$ is a frequency of j[th] path element in a i[th] document and it is multiplied by level weight $\frac{1}{x^n}$ in order to consider the semantic importance of a path element in a document. *x* refers to the level number of the highest tag of a tag path. The level number of the root tag is 1. That of a tag under the root tag is 2. And so on. *n* is a real number larger than 1. In this paper, 1 is chosen for the value of n. $PEIDF_{ij}$ is shown in equation (3).

$$PEIDF_{ij} = \log \frac{N}{DF_j} \qquad (3)$$

where, N be the total number of documents and DF_j be the number of documents in which the j[th] path element appears. Table 1 shows PEWF, PEIDF, and PESSW on an example trees in Figure 2.

Table 1. An example of *PTWF, PTIDF* and *PESSW*

	PEWF			PEIDF			PESSw		
	doc1	doc2	doc3	doc1	doc2	doc3	doc1	doc2	doc3
/B/I/T/D	1.00	0.00	0.00	1.10	1.10	1.10	1.10	0.00	0.00
/B/I/A/F	1.00	1.00	0.00	0.41	0.41	0.41	0.41	0.41	0.00
/B/I/A/L	1.00	1.00	0.00	0.41	0.41	0.41	0.41	0.41	0.00
/M/I/A/L	0.00	0.00	1.00	1.10	1.10	1.10	0.00	0.00	1.10
/B/I/T/~	1.00	0.00	0.00	1.10	1.10	1.10	1.10	0.00	0.00
/B/I/A/~	2.00	2.00	0.00	0.41	0.41	0.41	0.82	0.82	0.00
/M/I/A/~	0.00	0.00	1.00	1.10	1.10	1.10	0.00	0.00	1.10
/B/I/~/~	3.00	2.00	0.00	0.41	0.41	0.41	1.23	0.82	0.00
/M/I/~/~	0.00	0.00	1.00	1.10	1.10	1.10	0.00	0.00	1.10
/B/~/~/~	3.00	2.00	0.00	0.41	0.41	0.41	1.23	0.82	0.00
/M/~/~/~	0.00	0.00	1.00	1.10	1.10	1.10	0.00	0.00	1.10
/B/I/T	0.00	1.00	0.00	1.10	1.10	1.10	0.00	1.10	0.00
/M/I/T	0.00	0.00	1.00	1.10	1.10	1.10	0.00	0.00	1.10
/B/I/~	0.00	1.00	0.00	1.10	1.10	1.10	0.00	1.10	0.00
/M/I/~	0.00	0.00	1.00	1.10	1.10	1.10	0.00	0.00	1.10
/B/~/~	0.00	1.00	0.00	1.10	1.10	1.10	0.00	1.10	0.00
/M/~/~	0.00	0.00	1.00	1.10	1.10	1.10	0.00	0.00	1.10
/~/I/T/D	0.50	0.00	0.00	1.10	1.10	1.10	0.55	0.00	0.00
/~/I/A/F	0.50	0.50	0.00	0.41	0.41	0.41	0.20	0.20	0.00
/~/I/A/L	0.50	0.50	0.50	0.00	0.00	0.00	0.00	0.00	0.00
/~/I/T/~	0.50	0.00	0.00	1.10	1.10	1.10	0.55	0.00	0.00
/~/I/A/~	1.00	1.00	0.50	0.00	0.00	0.00	0.00	0.00	0.00
/~/I/~/~	1.50	1.00	0.50	0.00	0.00	0.00	0.00	0.00	0.00
/~/I/T	0.00	0.50	0.50	0.41	0.41	0.41	0.00	0.20	0.20
/~/I/~	0.00	0.50	0.50	0.41	0.41	0.41	0.00	0.20	0.20
/~/~/T/D	0.33	0.00	0.00	1.10	1.10	1.10	0.36	0.00	0.00
/~/~/A/F	0.33	0.33	0.00	0.41	0.41	0.41	0.14	0.14	0.00
/~/~/A/L	0.33	0.33	0.33	0.00	0.00	0.00	0.00	0.00	0.00
/~/~/T/~	0.33	0.00	0.00	1.10	1.10	1.10	0.36	0.00	0.00
/~/~/A/~	0.67	0.67	0.33	0.00	0.00	0.00	0.00	0.00	0.00
/~/~/T	0.00	0.33	0.33	0.41	0.41	0.41	0.00	0.14	0.14
/~/~/~/D	0.25	0.00	0.00	1.10	1.10	1.10	0.28	0.00	0.00
/~/~/~/F	0.25	0.25	0.00	0.41	0.41	0.41	0.10	0.10	0.00
/~/~/~/L	0.25	0.25	0.25	0.00	0.00	0.00	0.00	0.00	0.00

2.3. Integrating path element semantics vector model into SVD_{LSI}

Vector Space Model (VSM) [8] has long been used to represent unstructured documents as document feature vectors containing element occurrence statistics. By taking the vector space approach for representing XML documents, PEVSM as described previous section, inherits the limitation of VSM—terms are assumed to be independent of each other. Lacking the capability to represent terms' semantic relationships could result in problematic cases caused by polysemies and synonyms. Latent Semantic Indexing (LSI) [9] is a technique commonly used in information retrieval for overcoming the aforementioned problems caused by synonyms and polysemies. In particular, LSI projects a document from the original document feature space onto a corresponding "semantic" space via singular value decomposition (SVD) so that more robust semantic-based document similarity measure can be resulted. Using LSI, the original path element document matrix $PESSW_{mxn}$ is first decomposition into three matrices:

$$PESSW_{mxn} = U_{mxm} \cdot S_{mxn} \cdot V_{nxn}^T$$

where U and V contain orthonormal columns and S is diagonal. By restricting the matrices U, V and S to their first k <min(m,n) columns, one obtains the matrix

$$\hat{D}_{mxn}(PESSW_{mxn}) = \hat{U}_{mxk} \cdot \hat{S}_{kxk} \cdot \hat{V}_{kxn}^T$$

where $\hat{D}$ is the best square approximation of D by a matrix of rank k [9]. The newly defined path element document matrix will contain document feature vectors with path element semantics (obtained based on path element co-occurrence statistics) taken into consideration. To deal with novel documents not included in the path element-document matrix D, one can project the novel document vector onto the "semantic space" of dimension k and measure distance directly in the semantic space. According to [9], a novel document d's projection can be computed as:

$$d_{LSI}^T = d^T \cdot U \cdot S^{-1}$$

where $U \cdot S^{-1}$ is the transformation for the projection. Another alternative is to use simply U and the corresponding pseudo document projection becomes

$$d_{LSI}^T = d^T \cdot U$$

which is equivalent to put $d_{LSI}^T = d^T \cdot U \cdot S^0$.

To apply SVD_{LSI} to PESSW, XML documents are first partitioned into segments based on the element tags. SVD_{LSI} is then applied to the segment-path element matrix. Thus, an XML document will eventually be represented as a matrix $d_x \in R^{kxm}$, with each column being the projection of the element-specific feature vector on the semantic space. The rationale is that each XML element instance should be a semantically self-contained unit. We call this version of PESSW as PESSW-SVD_{LSI} in the subsequent sections.

3. Incremental SVD Managing Dynamic Collections

Our new folding-updating method uses a combination of folding-in and updating to modify the thin-SVD of the path-document matrix. The method determines when to update based on the number of documents that have been folded-in, relative to the size of the initial path-document matrix, or to the size of the last updated matrix if updates have already taken place. The method begins by folding-in documents, but only until the number of documents folded-in reaches a predetermined percentage *p* of the documents in the original matrix. The changes that have been made to matrix V_k by the folding-in process are then discarded, and the thin-SVD is updated using the updating methods of Zha and Simon [4]. Folding-in is then resumed until the number of new documents folded-in reaches a predetermined percentage *p* of the updated matrix, and so on. This process requires saving the document vectors that have been folded-in between updates, but repays this overhead with faster computation times than updating alone, and better accuracy than folding-in alone.

3.1. Folding-In. Folding a new document vector into the column space of an existing path-by-document matrix amounts to finding coordinates for that document in the basis U_k. The first step in folding a new $t \times 1$ document vector $\hat{d}_p$ into the column space is to project it onto that space. Let $\tilde{d}_p$ represent the projection of $\hat{d}_p$; then,

$$\tilde{d}_p = U_k U_k^T \hat{d}_p \qquad (4)$$

This equation shows that the coordinates of $\tilde{d}_p$ in the basis U_k are given by the elements of the vector $U_k^T \hat{d}_p$. The new document is then folded in by appending the *k*-dimensional vector $U_k^T \hat{d}_p$ as a new column of $k \times d$ the matrix $S_k V_k^T$. Because the latter matrix product is not actually computed, the folding-in is carried out implicitly by appending $\hat{d}_p{}^T U_k S_k^{-1}$ as a new row of V_k to form a new matrix $V_k^{'}$. The implicit product $S_k V_k^{'}$ is then the desired result. Note that the matrix $V_k^{'}$ is no longer orthonormal. In addition, the row space of the matrix $V_k^{'T}$ does not represent the row space of the new path-by-document matrix. Furthermore, if the new document $\tilde{d}_p$ is nearly orthogonal to the columns of U_k, most information about that document is lost in the projection step. Our proposed folding document describe as the following algorithm.

Algorithm 1 Algorithm for SVD_{LSI}-Folding Documents

/* Let $\hat{d}_p \in \Re^{m \times d}$ be the d new documents that should be folded to the existing document at the right of the old path-document matrix. */

1: Input: k,
$U_k \in \Re^{m \times k}, S_k \in \Re^{k \times k}, V_k \in \Re^{n \times k}, \hat{d}_p \in \Re^{m \times d}$.

2: Compute the projection: $\tilde{d}_p = \hat{d}_p U_k U_k^T \in \Re^{m \times d}$.

3: Compute the $d_k = \hat{d}_p^T U_k S_k^{-1}$, where $d_k \in \Re^{m \times d}$.

4: Append the $\hat{d}_p^T U_k S_k^{-1}$ as a new row of V_k to form a new matrix $V_k^{'}$ where new matrix $V_k^{'} \in \Re^{(d+d) \times d}$.

5: Output: The best rank-k approximation of
$E_{\Delta d} = (D_k, \hat{d}_p)$ is given by:

$E_k^{\Delta d} = U_k S_k V_k^{'T}$,

where $U_k \in \Re^{m \times k}, S_k \in \Re^{k \times k}, V_k^{'T} \in \Re^{k \times (n+d)}$.

Note that U_k, S_k are unchanged and $V_k^{'}$ is no longer orthonormal.

3.2. SVD-Updating. An alternative to folding-in that accounts for the effects that new terms and documents might have on term-document associations while still maintaining orthogonality was first described in [3] and [5]. This approach comprises the following three steps: updating terms, updating documents, and updating term weights. As pointed out by Simon and Zha [3], the operations discussed in [3] and [5] may not produce the exact SVD of the modified reduced-rank LSI model. Those authors provide alternative algorithms for all three steps of SVD-updating, and we now review them. For consistency with our earlier discussion, we use column pivoting in the QR factorizations, although it is not used in [3], [5], and [4]. Our proposed SVD-Updating document describes as the following algorithm.

Algorithm 2 Algorithm for SVD_{LSI}-Updating Documents

/* Let $\hat{d}_p \in \Re^{m \times d}$ be the d new documents that should be added to the existing document at the right of the old path-document matrix. */

1: Input: k,
$U_k \in \Re^{m\times k}, S_k \in \Re^{k\times k}, V_k \in \Re^{n\times k}, \hat{d}_p \in \Re^{m\times d}$.

2: Compute the projection:
$\tilde{d}_p = (I_m - U_k U_k^T)\hat{d}_p \in \Re^{m\times d}$.

3: Compute the QR decomposition: $\tilde{d}_p = \hat{Q}_d \hat{R}_d$, where
$\hat{Q}_d \in \Re^{m\times d}, \hat{R}_p \in \Re^{d\times d}$.

4: Compute the SVD of matrix

$$\hat{\alpha} \equiv \begin{pmatrix} S_k & U_k^T \hat{d}_p \\ 0_{d\times k} & \hat{R}_d \end{pmatrix} \in \Re^{(k+d)\times(k+d)}$$

in the form:

$$\hat{\alpha} = (u_k, u_k^T) \cdot diag(\hat{\sigma}_k, \hat{\sigma}_d) \cdot (v_k, v_k^T),$$

where $u_k, v_k \in \Re^{(k+d)\times k}$ and $\hat{\sigma}_k \in \Re^{k\times k}$.

5: Output: The best rank-k approximation of
$E_{\Delta d} = (D_k, \hat{d}_p)$ is given by:

$$E_k^{\Delta d} = [(U_k, \hat{Q}_d)u_k] \cdot \hat{\sigma}_k \cdot \left[\begin{pmatrix} V_k & 0_{n\times d} \\ 0_{d\times k} & I_d \end{pmatrix} v_k \right]^T, \text{ where}$$

$$U^{\Delta d} = (U_k, \hat{Q}_d)u_k, \text{ and } V^{\Delta d} = \begin{pmatrix} V_k & 0 \\ 0 & I_d \end{pmatrix} v_k, \; S_k = \hat{\sigma}_k.$$

4. Experimental Evaluation

Examples for this paper are run using Rstudio/R i386 4.1.2 on a Window 10. These results are produced using the real sets of the XML documents. Based upon these sets of XML documents with dynamic updating characteristics, their accuracy of information retrieval were computed, analyzed and reported as follows. The measure of similarity is the cosine of the angle between the query and document vectors. The path-document matrix D_{PESSW} is partitioned such that the first 200 columns form the initial matrix, and the remaining 600 columns are added incrementally. In each example, the average precision for each of the queries at the eleven standard recall levels (0%, 10%, . . . , 100%) is averaged to produce the overall average precision at each increment. The average precisions for the four methods discussed in this paper (re-computing, folding-updating, updating, and folding-in) are compared. For each example, k = 120, where k is the number of singular values and corresponding left and right singular vectors computed, and p = 10; when the folding-up method has folded-in documents equal to 10% of the initial path-document matrix, the thin-SVD is updated, and then folding-in resumes until the number of new documents folded-in reaches 10% of the updated matrix, and so on.

4.1 Working on Real Data Sets

The following five DTDs were downloaded from ACM's SIGMOD Record homepage [17]: 300 XML documents from OrdinaryIssuePage.dtd (O in short), ProceedingsPage1999.dtd (P-1999 in short), ProceedingsPage2002.dtd (P-2002 in short), IndexTerm1999.dtd (IT-1999 in short), Ordinary2002.dtd (Ord-2002 in short) and Ordinary2005.dtd (Ord-2005 in short). For another real data set we used the documents on ADC/NASA [18]:150 XML documents from adml.dtd (Astronomical Dataset Markup Language DTD). Also we download the nigara data [18]: 150 XML documents from movie.dtd, department.dtd, club.dtd, and personnel.dtd. Based upon these sets of XML documents with their precision for four methods, re-computing, folding-updating, updating, and folding-in, were computed, analyzed and reported as follows.

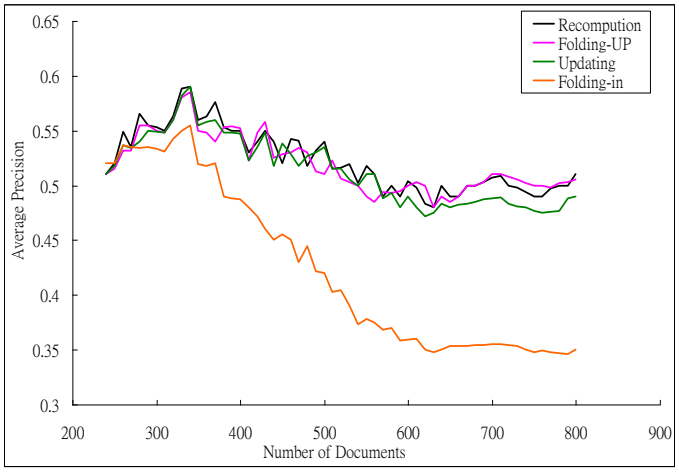
In the first example (see Figure 5), the initial path-document matrix of 862 terms and 200 documents has 600 documents added to it in 60 increments of 10 documents each, simulating a dynamic environment in which frequent small changes are made to the path-document matrix. Note that the initial matrix more than doubles in size as a result of the incremental additions. As expected, Figure 5 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 then the deterioration is very slight. The folding-up method gives similar results to re-computing the thin-SVD at every increment, but as Table 2 illustrates, in this example the folding-up method is more than 400 times faster than re-computing. The folding-up method gives even better results than the updating method for much of Figure 5 and yet, in this example, it is more than three times faster than the updating method.

In the second example (see Figure 6), the initial path-document matrix with 862 terms and 200 documents once again has 600 documents added to it, but in this case there are 30 increments of 20 documents each. As in the first example, this simulates a dynamic environment in which the path-document matrix is enlarged frequently. As in the previous example, Figure 6 shows that the average precision for the folding-in method deteriorates rapidly compared to the other methods. In this example both the

updating method and the folding-updating method give similar results when compared to the method of re-computing at each increment, but the updating method is more than 120 times faster than re-computing, and the folding-up method is more than 200 times faster than re-computing.

Table 2: CPU times (seconds): 600 documents added in 10 and 20.

Method	CPU time	CPU time
	Increments of 60	Increments of 30
Recomputing	2537.84	1341.55
Folding-update	6.25	6.05
Updating	19.63	10.66
Folding	0.61	0.34



FFigure 5: Average precisions for four methods using DPESSW: 600 documents added in 60 increments of 10.

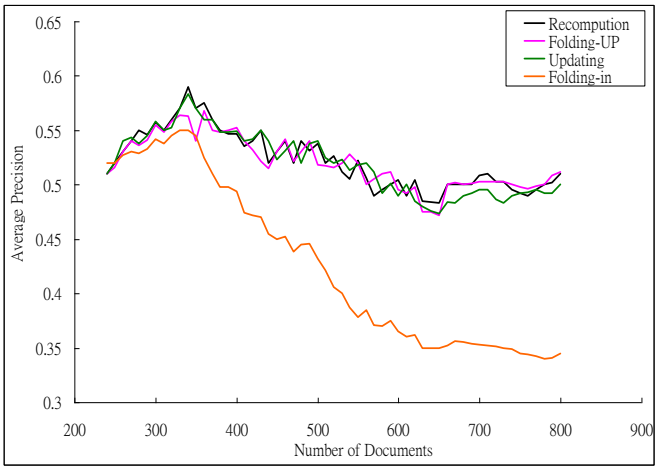


Figure 6: Average precisions for four methods using DPESSW: 600 documents add

5. Conclusion

The experiment result is two-fold. First, we have demonstrated that the updating methods proposed by Zha and Simon [4] are effective in a dynamic environment in which there are many small updates made to the path-document matrix. This method of updating the thin-SVD achieves similar average precision to re-computing the thin-SVD, using only a fraction of the computation time. This in itself is significant, but we have also demonstrated that our new hybrid method, folding-updating, is an even more attractive option than updating alone. As with the updating method, the new folding-updating method achieves average precision similar to that of re-computing the thin-SVD, but the folding-up method requires less computation time than either re-computing or updating the thin-SVD. Next research issue will go through the efficient indexing method such as the R-tree in order to efficiently search interesting documents by user's request.

REFERENCES

[1] 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.

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

[3] 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.

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

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

[6] Berry M.W. and Dumasis S.T. (1995): Using linear algebra for intelligent information retrieval. — SIAM Rev., Vol. 37, No. 4, pp. 573–995.

[7] Berry M.W., Drmac Z. and Jessup E.R. (1999): Matrices, vector spaces, and information retrieval. — SIAM Rev., Vol. 41, No. 2, pp. 335–362.

[8] Gao J. and Zhang J. (2005): Clustered SVD strategies in latent semantic indexing.—Inf. Process. Manag., Vol. 41, No. 5, pp. 1051–1063.

[9] Ye Y.Q. (2000): Comparing matrix methods in text-based information retrieval. — Tech. Rep., School of Mathematical Sciences, Peking University.

[10] Gao J. and Zhang J. (2005): Clustered SVD strategies in latent semantic indexing.—Inf. Process. Manag., Vol. 41, No. 5, pp. 1051–1063.

[11] Landauer T.K., Foltz P.W. and Laham D. (1998): Introduction to latent semantic analysis. — Discourse Processes, Vol. 25, pp. 259–284.

[12] Balinski J. and Danilowicz C. (2005): Ranking method based on inter document distances.—Inf. Process. Manag., Vol. 41, No. 4, pp. 759–775.

[13] Berry M.W. and Shakhina A.P. (2005): Computing sparse reduced-rank approximation to sparse matrices. — ACM Trans. Math. Software, 2005, Vol. 31, No. 2, pp. 252–269.

[14] Gao J. and Zhang J. (2005): Clustered SVD strategies in latent semantic indexing.—Inf. Process. Manag., Vol. 41, No. 5, pp. 1051–1063.

[15] Bass D. and Behrens C. (2003): Distributed LSI: Scalable concept based information retrieval with high semantic resolution.—Proc. 2003 Text Mining Workshop, San Francisco, CA, USA, pp. 72–82.

[16] Fan J., Ravi K., Littman M.L. and Santosh V. (1999): Efficient singular value decomposition via document samplings. — Tech. Rep. CS-1999-5, Dept. Computer Science, Duke University, North Carolina.

[17] ACM SIGMOD Record home page [http://www.acm.org/sigmod/record/xml]

[18] http://www.cs.wisc.edu/niagara/data/

BIOGRAPHIES

Hsu-Kuang Chang He is associative professor in the Department of Information Engineering, I-Shou University, Taiwan. His research interests include data mining, multimedia media database, and information retrieval.