Latent Semantic Indexing for Image Retrieval Systems

***Pavel Praks*[*], *Jiří Dvorský*, *Václav Snášel*[†]**

1 Latent Semantic Indexing

Matrix computation is used as a basis for information retrieval in the retrieval strategy called Latent Semantic Indexing (LSI) [1]. The premise is that more conventional retrieval strategies (i.e. vector space, probabilistic and extended Boolean) all have problems because they match directly on keywords. Since the same concept can be described using many different keywords, this type of matching is prone to failure. The authors cite a study in which two people used the same word for the same concept only twenty percent of the time.

LSI tries to search for something that is closer to representing the underlying semantics of a document. The searching is done by using matrix computation, in particular Singular Value Decomposition (SVD). This filters out the noise found in a document, such that two documents that have same semantics will be located close to one another in a multi-dimensional space.

Majority of images in real world are stored as raster images. Image can be viewed as vector of pixels; every pixel is described by its color. The vector of pixel represents some kind of keywords in image. But human observer extracts from image important features that define semantics of image for him. The man doesn't think about pixel but about persons or objects on image. So we need technique that is able to extract this features and that is resistant to minor changes of images (e.g. amount of light, contrast and moves of objects on the images). Direct usage of keyword based systems leads to results that are sensitive to small change of any keyword (pixel in query).

[*]Dept. of Applied Mathematics, Technical University of Ostrava,17. listopadu 15, Ostrava - Poruba, Czech Republic, pavel.praks@vsb.cz

[†]Department of Computer Science, Technical University of Ostrava, 17. listopadu 15, Ostrava - Poruba, Czech Republic, jiri.dvorsky,vaclav.snasel@vsb.cz

2 Numerical aspects of Latent Semantic Indexing

Let the symbol A denotes the $m \times n$ term-document matrix related to m dictionary terms in n documents. Let us remind the (i, j) element of the term-document matrix A represents the number of occurence the i-th term in the j-th document. The matrix A is often sparse, because each document usually does not contain all dictionary terms.

The LSI procedure involves a *Singular value decomposition* (SVD) of the term-document matrix A. The aim of SVD is to compute decomposition

$$A = USV^T, \tag{1}$$

where $S \in R^{m \times n}$ is a diagonal matrix with nonnegative diagonal elements called the singular values, $U \in R^{m \times m}$ and $V \in R^{n \times n}$ are orthogonal matrices[1]. The columns of matrices U and V are called the left singular vectors and the right singular vectors respectively. The decomposition can be computed so that the singular values are sorted by decreasing order.

The full SVD decomposition (2) is memory and time consuming operation, especially for large problems. Although the document matrix A is often sparse, the matrices U and V have dense structure. Due this facts, only a few largest singular values of A and the corresponding left and right singular vectors are computed and stored in memory. The number of singular values and vectors which are computed and keeped in memory can be choosen as a compromise between the speed/precision ratio of the LSI procedure.

3 Fast LSI Algorithm

To eliminate big time complexity of SVD new method was developed which can perform LSI analysis without SVD. In this case it is satisfactory to compute only several eigenvectors and eigenvalues of square symmetric positive definitive matrix. There are more efficient algorithms for such kind of matrix. Moreover the dimension of decomposed matrix is equal to number of documents, the dimension doesn't depends on number of terms in collection. Experimental results show, that the algorithm is faster with growing size of decomposed matrix (compared to SVD).

4 The software implementation of Latent Semantic Indexing

We implemented and tested LSI procedure in the Matlab system by Mathworks. The term-document matrix A was decomposed by the Matlab command `svds`. Using the `svds` command brings following advantages:

- The term-document matrix A can be effectively stored in memory by using the Matlab storage format for sparse matrices.

[1] A matrix $Q \in R^{n \times n}$ is said to be *orthogonal* if the condition $Q^{-1} = Q^T$ holds.

- The number of eigenvalues and eigenvectors computed by the LSI can be easily set by user.

Following [2] the Latent Semantic Indexing procedure can be writtten in Matlab as follows.

Procedure LSI [Latent Semantic Indexing]

```
function sim = lsi(A,q,k)

% Input:
% A ...  the m × n document matrix
% q ...  the query vector
% k ...  Compute k largest eigenvalues and eigenvectors; k ≤ n

% Output:
% sim ...  the vector of similarity coefficients

[m,n] = size(A);
```

1. **Compute the co-ordinates of all documents in the k-dimensional space by the LSI of the term-document matrix A.**

   `[U,S,V] = eigenv(`$A^T \times A$`,k);`
   % Compute the k eigenvalues of A;

2. **Compute the co-ordinate of the query vector q**

   ```
   qc = q' * U * inv(S);
   ```
   % The vector qc includes the co-ordinate of the query vector q; The matrix $inv(S)$ contains reciprocals of singular values; The symbol ' denotes the transpose superscript.

3. **Compute the similarity coefficients between the co-ordinates of the query vector and documents.**

   ```
   for i = 1:n      % Loop over all documents
         sim(i) = (qc * V(i,:)')  / (norm(qc) * norm(V(i,:)));
   end;
   ```
   % Compute the similarity coefficient for i-th document; $V(i,:)$ denotes the i-th row of V.

The procedure `lsi` returns to the user the vector of similarity coefficients sim. The i-th element of the vector sim contains a value which indicate a measure of the semantic similarity between the i-th document and the query document. The increasing value of the similarity coefficient indicates the increasing semantic similarity.

Query: Pic5	
Picture	Similarity
Pic5	1.0000
Pic2	0.9740
Pic6	0.3011
Pic4	0.2242
Pic24	0.1697
Pic3	0.1260

Table 1. *Similarity of pictures for Query1*

Query: Pic7	
Picture	Similarity
Pic7	1.0000
Pic8	0.9769
Pic9	0.9165
Pic23	0.1345
Pic14	0.1011
Pic31	0.1007
Pic25	0.0847
Pic19	0.0752
Pic6	0.0672

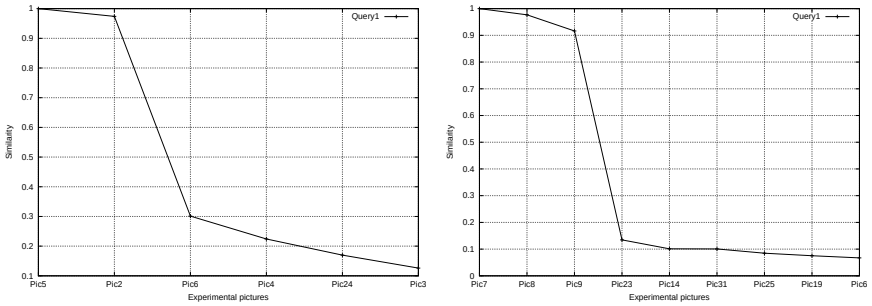
Table 2. *Similarity of pictures for Query2*

5 Experimental results

Experimental implementation was written in MatLab to test our algorithm. Collection of 100 grayscale images was processed. Width of images was 640 pixels, height 480 pixels (e.g. there are 307200 "keywords"). The queries were represented as images from the collection. The most time consuming part of LSI was multiplying of matrices $A^T \times A$. It takes on Pentium III , 512 MB RAM running Debian Linux 1807 seconds. The computation of eigenvalues takes only 3 seconds.

5.1 Query1

The first query was image Pic5 (see Fig 2) (river with a part of boat on the right side). As can seen in table 1 the most similar is (of course) Pic5 itself. The second selected image was Pic2 (the same river but without boat). The Pic6 and Pic4 are the third and the fourth selected images. As you can see the boat moved to the left and another man appears. The boat cross the waves and appears in the bright part of image. Pic24 is completely mismatched.



(a) Query1 (b) Query2

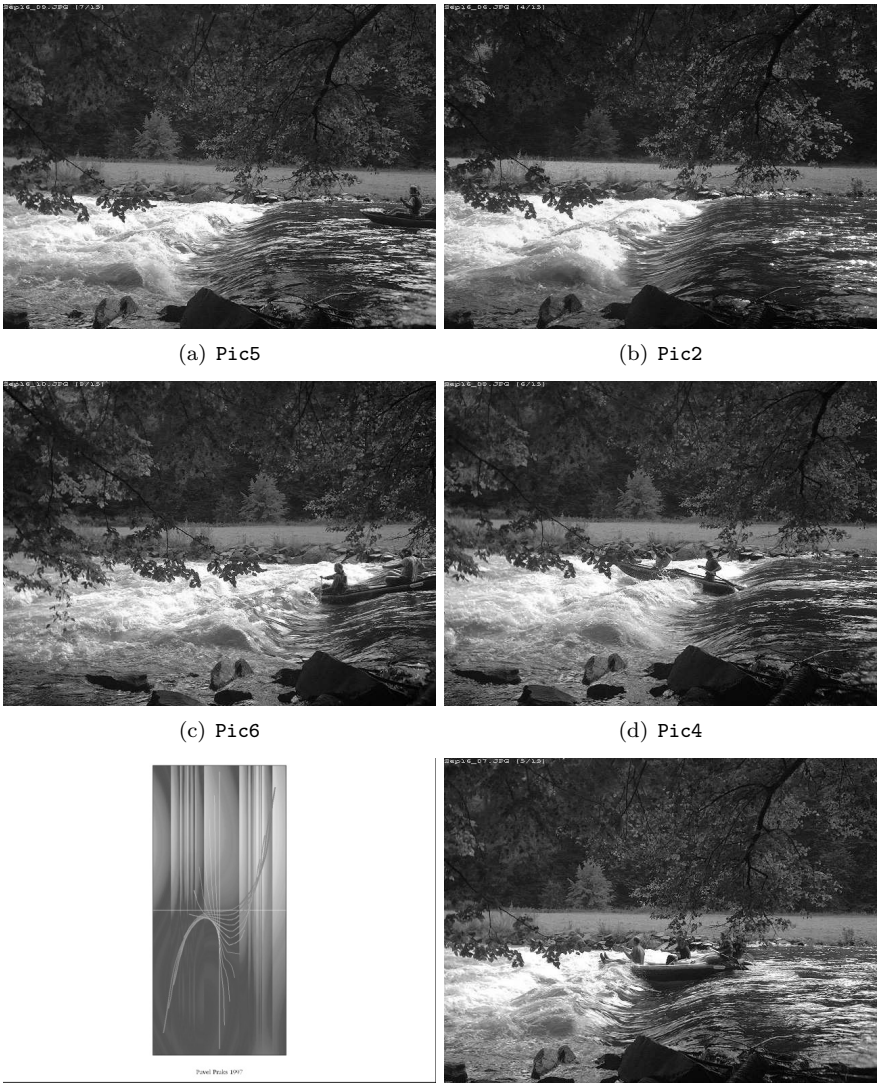
Figure 1. *Similarity of pictures for Query1 and Query2*

5.2 Query2

The second query was image Pic7 (see Fig 3) (presents from Pavel's birthday party). The picture itself is again he most similar (see table 2). The Pic8 is the same set of objects but the snapshot was taken from little different angle. A bottle is missing on the third selected image, but bar of chocolate is added. The next selected images have similarity ten times smaller than these three images and they are mismatched.

6 Conclusion

The new fast LSI algorithm was presented. The algorithm doesn't depend on dimension of keywords, pixel extracted from processed documents, images. The experiments prove that the algorithm is suitable for image retrieval. Test on text documents will be done in the near future. The implementation independent on Matlab will be considered and tests with sparse matrix with random access [3] will be done.



(a) `Pic5` (b) `Pic2`

(c) `Pic6` (d) `Pic4`

(e) `Pic24` (f) `Pic3`

Figure 2. *Pictures used in the Query1*

(a) `Pic7`

(b) `Pic8`

(c) `Pic9`

(d) `Pic23`

(e) `Pic14`

Figure 3. *Pictures used in the Query2*

Bibliography

[1] S. Deerwester, S. Dumais, G. Furnas, T. Landauer, R. Harshman, *Indexing by latent sematic analysis*, Journal of the American Society for Information Science, 41(6):391-407.

[2] D. A. Grossman, O. Frieder, *Information retrieval: Algrithms and heuristics*, Kluwer Academic Publishers, Second edition 2000

[3] V. Snášel, J. Dvorský, V. Vondrák, *Random access storage system for sparse matrices*, Proccedings of ITAT 2002, Brdo, High Fatra, Slovakia, 2002