

Fast LSI-based techniques for query expansion in text retrieval systems

L. Laura U. Nanni **F. Sarracco**

Department of Computer and System Science
University of Rome "La Sapienza"

2nd Workshop on Text-based Information Retrieval
Koblenz, 11 Sept. 2005

Outline

- 1 Preliminaries
 - Classical Text Matching
 - Query expansion by Thesauri
 - Spectral Techniques
- 2 Our approaches
 - LS-Thesaurus
 - LS-Filter
- 3 Experimental results
- 4 Conclusions

Term-Documents matrix

Collection $D = \{d_1, \dots, d_n\}$ of text documents.

$T = \{t_1, \dots, t_m\}$: set of distinct index terms in D :

$$A = \begin{matrix} & & d_1 & d_2 & \cdots & d_n \\ \begin{matrix} t_1 \\ t_2 \\ \vdots \\ t_m \end{matrix} & \left(\begin{array}{cccc} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \cdots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{array} \right) \end{matrix}$$

$a_{i,j}$ is a function of the weight of term t_i in document d_j

Term-Documents matrix

Collection $D = \{d_1, \dots, d_n\}$ of text documents.

$T = \{t_1, \dots, t_m\}$: set of distinct index terms in D :

$$A = \begin{matrix} & & d_1 & d_2 & \cdots & d_n \\ \begin{matrix} t_1 \\ t_2 \\ \vdots \\ t_m \end{matrix} & \left(\begin{matrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \cdots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{matrix} \right) \end{matrix}$$

$a_{i,j}$ is a function of the weight of term t_i in document d_j

Term-Documents matrix

Collection $D = \{d_1, \dots, d_n\}$ of text documents.

$T = \{t_1, \dots, t_m\}$: set of distinct index terms in D :

$$A = \begin{matrix} & & d_1 & d_2 & \cdots & d_n \\ \begin{matrix} t_1 \\ t_2 \\ \vdots \\ t_m \end{matrix} & \left(\begin{matrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \cdots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{matrix} \right) \end{matrix}$$

$a_{i,j}$ is a function of the weight of term t_i in document d_j

Term-Documents matrix

Collection $D = \{d_1, \dots, d_n\}$ of text documents.

$T = \{t_1, \dots, t_m\}$: set of distinct index terms in D :

$$A = \begin{matrix} & & d_1 & d_2 & \cdots & d_n \\ \begin{matrix} t_1 \\ t_2 \\ \vdots \\ t_m \end{matrix} & \left(\begin{matrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \cdots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{matrix} \right) \end{matrix}$$

$a_{i,j}$ is a function of the weight of term t_i in document d_j

TF-IDF term-weighting schemes

The value of $a_{i,j}$ is a function of two factors

A LOCAL FACTOR $L(i, j)$

measuring the relevance of term t_i in document d_j .

We used:

$$L(i, j) = \frac{\text{freq}(i, j)}{\max_{i \in [1, m]} \text{freq}(i, j)}$$

A GLOBAL FACTOR $G(i)$

to de-amplify the relative weight of terms which are very frequently used in the collection.

We used:

$$G(i) = \log \frac{n}{n_i}$$

n_i is the number of documents containing term t_i .

TF-IDF term-weighting schemes

The value of $a_{i,j}$ is a function of two factors

A LOCAL FACTOR $L(i, j)$

measuring the relevance of term t_i in document d_j .

We used:

$$L(i, j) = \frac{\text{freq}(i, j)}{\max_{i \in [1, m]} \text{freq}(i, j)}$$

A GLOBAL FACTOR $G(i)$

to de-amplify the relative weight of terms which are very frequently used in the collection.

We used:

$$G(i) = \log \frac{n}{n_i}$$

n_i is the number of documents containing term t_i .

TF-IDF term-weighting schemes

The value of $a_{i,j}$ is a function of two factors

A LOCAL FACTOR $L(i, j)$

measuring the relevance of term t_i in document d_j .

We used:

$$L(i, j) = \frac{\text{freq}(i, j)}{\max_{i \in [1, m]} \text{freq}(i, j)}$$

A GLOBAL FACTOR $G(i)$

to de-amplify the relative weight of terms which are very frequently used in the collection.

We used:

$$G(i) = \log \frac{n}{n_i}$$

n_i is the number of documents containing term t_i .

Text Matching Algorithm

- **Input:** a query vector $\vec{q} = \{q_1, \dots, q_m\}$
- **Output:** the rank vector $\vec{r} = \vec{q} \cdot A$
- A and q are sparse $\Rightarrow \vec{q} \cdot A$ can be computed very efficiently.
- If the size of the query is limited to k terms \Rightarrow TM has cost $O(kn)$.

Issues

- *Polysemy*: e.g., polo
- *Synonymy*: e.g., automobile, car, machine

Text Matching Algorithm

- **Input:** a query vector $\vec{q} = \{q_1, \dots, q_m\}$
- **Output:** the rank vector $\vec{r} = \vec{q} \cdot A$
- A and q are sparse $\Rightarrow \vec{q} \cdot A$ can be computed very efficiently.
- If the size of the query is limited to k terms \Rightarrow TM has cost $O(kn)$.

Issues

- *Polysemy*: e.g., polo
- *Synonymy*: e.g., automobile, car, machine

Text Matching Algorithm

- **Input:** a query vector $\vec{q} = \{q_1, \dots, q_m\}$
- **Output:** the rank vector $\vec{r} = \vec{q} \cdot A$
- A and q are sparse $\Rightarrow \vec{q} \cdot A$ can be computed very efficiently.
- If the size of the query is limited to k terms \Rightarrow TM has cost $O(kn)$.

Issues

- *Polysemy*: e.g., polo
- *Synonymy*: e.g., automobile, car, machine

Text Matching Algorithm

- **Input:** a query vector $\vec{q} = \{q_1, \dots, q_m\}$
- **Output:** the rank vector $\vec{r} = \vec{q} \cdot A$
- A and q are sparse $\Rightarrow \vec{q} \cdot A$ can be computed very efficiently.
- If the size of the query is limited to k terms \Rightarrow TM has cost $O(kn)$.

Issues

- *Polysemy*: e.g., polo
- *Synonymy*: e.g., automobile, car, machine

Text Matching Algorithm

- **Input:** a query vector $\vec{q} = \{q_1, \dots, q_m\}$
- **Output:** the rank vector $\vec{r} = \vec{q} \cdot A$
- A and q are sparse $\Rightarrow \vec{q} \cdot A$ can be computed very efficiently.
- If the size of the query is limited to k terms \Rightarrow TM has cost $O(kn)$.

Issues

- *Polysemy*: e.g., polo
- *Synonymy*: e.g., automobile, car, machine

Query expansion

QUERY EXPANSION (or QUERY REWEIGHTING)

The process aimed to alter the weights, and possibly the terms, of a query.

- Two approaches:
 - 1 use relevance feedback
 - 2 use some knowledge on terms relationship (thesaurus)
- The term-term correlation matrix AA^T gives a statistic estimation of relationships among terms in the collection
⇒ Query expansion: $\vec{q}' \leftarrow \vec{q}AA^T$

Query expansion

QUERY EXPANSION (or QUERY REWEIGHTING)

The process aimed to alter the weights, and possibly the terms, of a query.

- Two approaches:
 - 1 use relevance feedback
 - 2 use some knowledge on terms relationship (thesaurus)
- The term-term correlation matrix AA^T gives a statistic estimation of relationships among terms in the collection
⇒ Query expansion: $\vec{q}' \leftarrow \vec{q}AA^T$

Query expansion

QUERY EXPANSION (or QUERY REWEIGHTING)

The process aimed to alter the weights, and possibly the terms, of a query.

- Two approaches:
 - 1 use relevance feedback
 - 2 use some knowledge on terms relationship (thesaurus)
- The term-term correlation matrix AA^T gives a statistic estimation of relationships among terms in the collection
⇒ Query expansion: $\vec{q}' \leftarrow \vec{q}AA^T$

Similarity Thesaurus

- Qiu and Frei presented an alternative approach to compute A
- Idea: compute the probability that a document is representative of a term
- They propose the following weighting scheme:

$$a_{i,j} = \begin{cases} \frac{(0.5 + 0.5 * \frac{freq(i,j)}{\max_j freq(i,j)}) * itf(j)}{\sqrt{\sum_{l=1}^n ((0.5 + 0.5 * \frac{freq(i,l)}{\max_l freq(i,l)}) * itf(l))^2}} & \text{if } freq(i,j) > 0 \\ 0 & \text{otherwise} \end{cases}$$

- $\max_j freq(i,j)$: is the maximum frequency of term t_i over all the documents in the collection.
- $itf_j = \log \frac{m}{m_j}$, m_j the number of distinct terms in the document d_j ;

Query expansion by Similarity Thesaurus

Algorithm

- 1: Compute \bar{A} with the previous weighting function;
- 2: Compute similarity thesaurus: $S \leftarrow \bar{A}\bar{A}^T$;
- 3: Given a query vector \vec{q} :
- 4: $\vec{s} \leftarrow \vec{q} S$;
- 5: $\vec{s}' \leftarrow \xi(\vec{s}, x_r)$;
- 6: $\vec{s}'' \leftarrow \frac{\vec{s}'}{|\vec{q}|}$, where $|q| = \sum_{i=1}^m q_i$;
- 7: $\vec{q}' \leftarrow \vec{q} + \vec{s}''$;

Search quality

Use of S.T. improves Text-Matching but does not completely resolve polysemy and synonymy.

Query expansion by Similarity Thesaurus

Algorithm

- 1: Compute \bar{A} with the previous weighting function;
- 2: Compute similarity thesaurus: $S \leftarrow \bar{A}\bar{A}^T$;
- 3: Given a query vector \vec{q} :
- 4: $\vec{s} \leftarrow \vec{q} S$;
- 5: $\vec{s}' \leftarrow \xi(\vec{s}, \mathbf{x}_r)$;
- 6: $\vec{s}'' \leftarrow \frac{\vec{s}'}{|\vec{q}|}$, where $|q| = \sum_{i=1}^m q_i$;
- 7: $\vec{q}' \leftarrow \vec{q} + \vec{s}''$;

Search quality

Use of S.T. improves Text-Matching but does not completely resolve polysemy and synonymy.

Latent Semantic Indexing

Idea

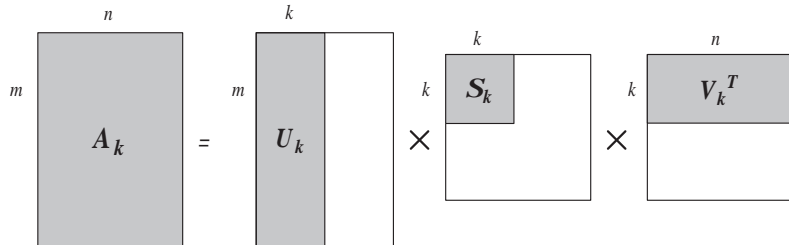
Use the Singular Value Decomposition to produce a low rank approximation of A

$$\begin{matrix} & n \\ & \boxed{A} \\ m & \end{matrix} = \begin{matrix} & r \\ & \boxed{U} \\ m & \end{matrix} \times \begin{matrix} & r \\ & \boxed{S} \\ r & \end{matrix} \times \begin{matrix} & n \\ & \boxed{V^T} \\ r & \end{matrix}$$

Latent Semantic Indexing

Idea

Use the Singular Value Decomposition to produce a low rank approximation of A



LSI Search

- The documents collection is represented in the reduced k -dimensional subspace:

$$D = \Sigma_k^{-1} U_k^T A$$

- Also the query vector is projected in the k -dimensional subspace: $q_k = \Sigma_k^{-1} U_k^T q$
- The rank of i -th document is given by

$$r_i = \cos \alpha_i = \frac{\vec{q}_k \cdot \vec{d}^i}{|\vec{q}_k| \cdot |\vec{d}^i|}$$

Issue

Requires the computation of n cosines \Rightarrow very slow, unusable for large collections

LSI Search

- The documents collection is represented in the reduced k -dimensional subspace:

$$D = \Sigma_k^{-1} U_k^T A$$

- Also the query vector is projected in the k -dimensional subspace: $q_k = \Sigma_k^{-1} U_k^T q$
- The rank of i -th document is given by

$$r_i = \cos \alpha_i = \frac{\vec{q}_k \cdot \vec{d}^i}{|\vec{q}_k| \cdot |\vec{d}^i|}$$

Issue

Requires the computation of n cosines \Rightarrow very slow, unusable for large collections

LSI Search

- The documents collection is represented in the reduced k -dimensional subspace:

$$D = \Sigma_k^{-1} U_k^T A$$

- Also the query vector is projected in the k -dimensional subspace: $q_k = \Sigma_k^{-1} U_k^T q$
- The rank of i -th document is given by

$$r_i = \cos \alpha_i = \frac{\vec{q}_k \cdot \vec{d}^i}{|\vec{q}_k| \cdot |\vec{d}^i|}$$

Issue

Requires the computation of n cosines \Rightarrow very slow, unusable for large collections

LSI Search

- The documents collection is represented in the reduced k -dimensional subspace:

$$D = \Sigma_k^{-1} U_k^T A$$

- Also the query vector is projected in the k -dimensional subspace: $q_k = \Sigma_k^{-1} U_k^T q$
- The rank of i -th document is given by

$$r_i = \cos \alpha_i = \frac{\vec{q}_k \cdot \vec{d}^i}{|\vec{q}_k| \cdot |\vec{d}^i|}$$

Issue

Requires the computation of n cosines \Rightarrow very slow, unusable for large collections

LS-Thesaurus

Idea

Compute the similarity matrix starting from a low rank approximation of \bar{A} (as in LSI).

We can formally define our similarity matrix S_k in the following way:

$$S_k = \bar{A}_k \bar{A}_k^T = \bar{U}_k \bar{\Sigma}_k \bar{V}_k^T \bar{V}_k \bar{\Sigma}_k^T \bar{U}_k^T = \bar{U}_k \bar{\Sigma}_k^2 \bar{U}_k^T$$

Note that \bar{U} and the diagonal elements of $\bar{\Sigma}$ correspond respectively to the eigenvectors and the eigenvalues of matrix $\bar{A}\bar{A}^T$.

Algorithm LS-Thesaurus - Pre-Process

- 1: **Input:** a collection of documents D ;
- 2: **Output:** a Similarity Thesaurus, i.e., a $m \times m$ matrix S_k ;
- 3:
- 4: Compute the term-document matrix \bar{A} from D ;
- 5: $(U, \Lambda) \leftarrow EIGEN(\bar{A}\bar{A}^T)$;
- 6: $U_k \leftarrow U^{(1:m, 1:k)}$;
- 7: $\Lambda_k \leftarrow \Lambda^{(1:k, 1:k)}$;
- 8: $S_k \leftarrow U_k \Lambda_k U_k^T$;

Algorithm LS-Thesaurus - Expand

- 1: **Input:** a query vector \vec{q} , a similarity thesaurus S_k ,
- 2: a positive integer x_r ;
- 3: **Output:** a query vector \vec{q}' ;
- 4:
- 5: $\vec{s} \leftarrow \vec{q} S_k$;
- 6: $\vec{s}' \leftarrow \xi(\vec{s}, x_r)$;
- 7: $\vec{s}'' \leftarrow \frac{\vec{s}'}{|\vec{q}|}$, where $|q| = \sum_{i=1}^m q_i$;
- 8: $\vec{q}' \leftarrow \vec{q} + \vec{s}''$;

LS-Filter

Assumptions

- In LSI algorithm documents and queries are projected (and compared) in a k -dimensional subspace
- The axes of this subspace represent the k most important concepts arising from the documents in the collection
- The user query tries to catch one (or more) of these concepts by using an appropriate set of terms

Idea

Let the system select the terms which best represent the required concepts

LS-Filter

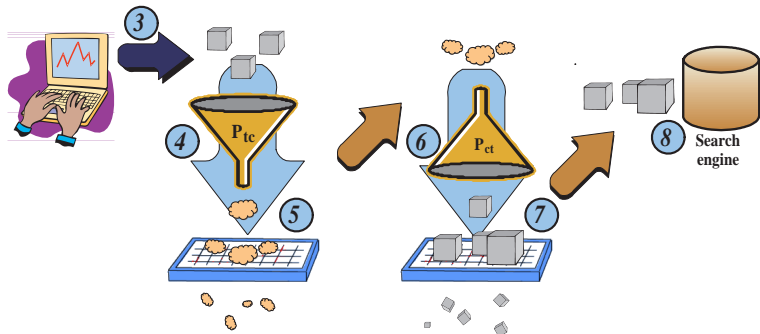
Assumptions

- In LSI algorithm documents and queries are projected (and compared) in a k -dimensional subspace
- The axes of this subspace represent the k most important concepts arising from the documents in the collection
- The user query tries to catch one (or more) of these concepts by using an appropriate set of terms

Idea

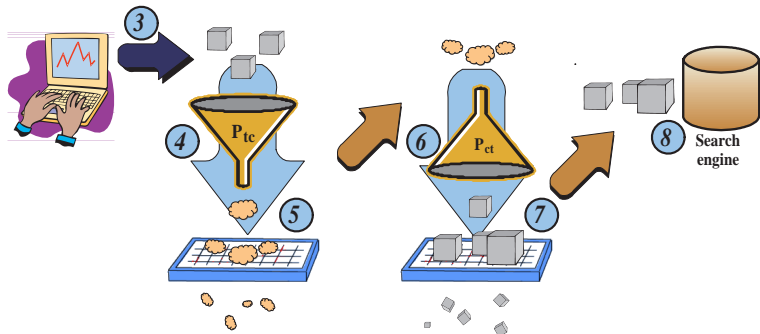
Let the system select the terms which best represent the required concepts

LS-Filter - Schematic view



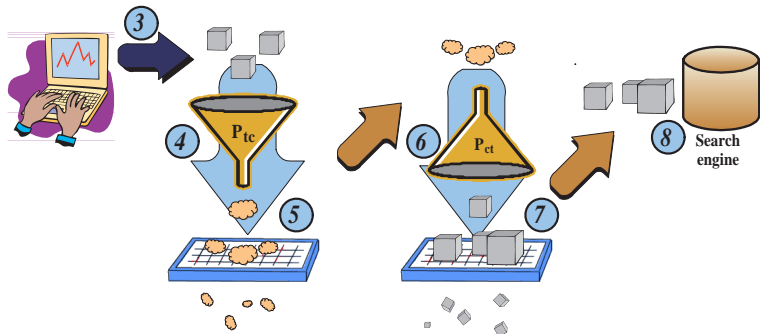
- 3. The user inserts a query

LS-Filter - Schematic view



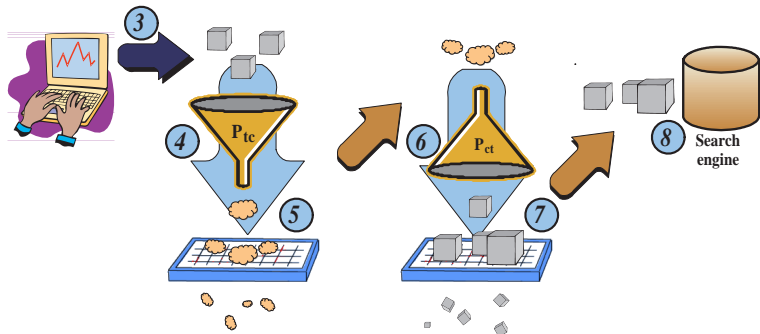
- 4. The query vector is projected in the concepts space

LS-Filter - Schematic view



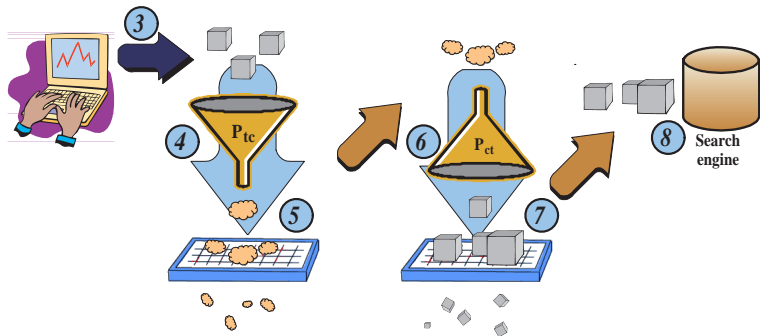
- 5. Minor concepts are removed

LS-Filter - Schematic view



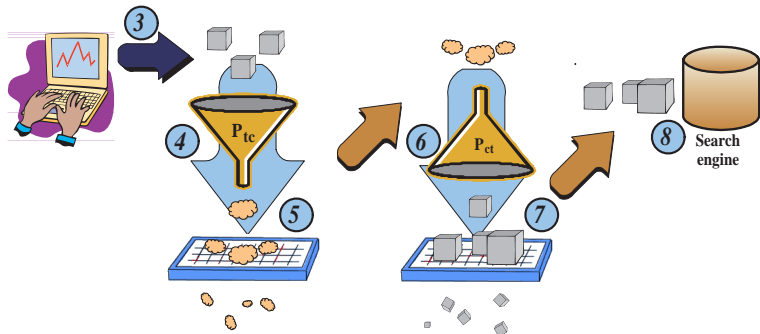
- 6. Concepts vector is projected back into the terms space

LS-Filter - Schematic view



- 7. Minor terms are removed

LS-Filter - Schematic view



- 8. Remaining terms are added to the original query vector

Algorithm LS-Filter - Pre-Process

- 1: **Input:** a collection of documents D ;
- 2: **Output:** a pair of matrices (P, P^{-1}) ;
- 3:
- 4: Compute the term-document matrix A from D ;
- 5: $(U, \Sigma, V) \leftarrow \text{SVD}(A)$;
- 6: $U_k \leftarrow U^{(1:m, 1:k)}$;
- 7: $\Sigma_k \leftarrow \Sigma^{(1:k, 1:k)}$;
- 8: $P \leftarrow \Sigma_k^{-1} U_k^T$;
- 9: $P^{-1} \leftarrow U_k \Sigma_k$;

Algorithm LS-Filter - Expand

- 1: **Input:** a query vector \vec{q} , matrices (P, P^{-1}) ,
- 2: two positive integers x_c and x_t ;
- 3: **Output:** a query vector \vec{q}' ;
- 4:
- 5: $\vec{p} \leftarrow P \vec{q}$;
- 6: $\vec{p}' \leftarrow \xi(\vec{p}, x_c)$;
- 7: $\vec{p}'' \leftarrow P^{-1} \vec{p}'$;
- 8: $\vec{q}' \leftarrow \xi(\vec{p}'', x_t)$;

Algorithms

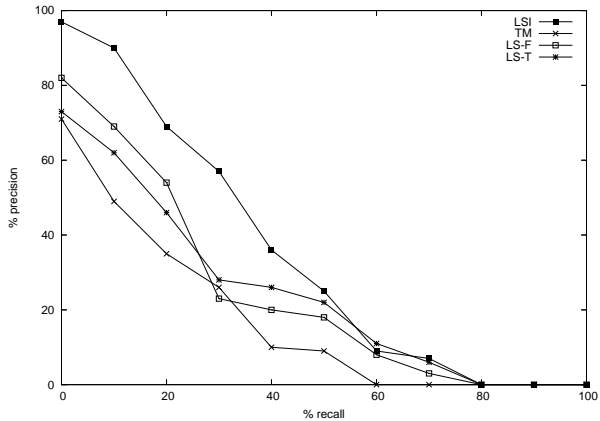
We compared the behavior of the following approaches:

- TM: the simple text matching;
- LS-T: text matching with queries previously expanded by *LS-Thesaurus* algorithm;
- LS-F: text matching with queries previously expanded by *LS-Filter* algorithm;
- LSI: the full *LSI* computation.

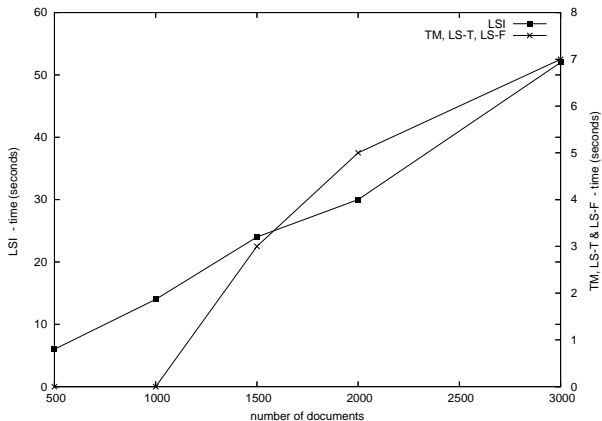
Dataset

- Three books from O'Reilly in html format about Perl, Unix and Java
- 3000 documents (html pages)
- 150 short queries, with human made collection of relevant documents
- publicly available on the web at URL:
`http://www.dis.uniroma1.it/~laura/`

Retrieval effectiveness



Time performances



Examples of LS-Filter

job control		shell logout		zip file	
job	0.42763	shell	0.68739	file	0.71548
background	0.12985	bourne	0.11818	util	0.16258
echo	0.10754	login	0.09497	zip	0.15056
number	0.09974	perl	0.09123	checksum	0.09776
list	0.07858	prompt	0.09851		
control	0.06929				
ctrl	0.06882				
object	0.06669				
line	0.05827				
filename	0.05827				

What we have done

- Previous techniques:
 - TM: fast but prone to synonymy and polysemy
 - LSI: effective but slow
- We introduced 2 techniques to improve TM search by using query expansion
- We compared them with TM and LSI search:
 - we used a non-standard mid-size dataset
 - generally their performances are better than TM, but not homogeneous

What we have done

- Previous techniques:
 - TM: fast but prone to synonymy and polysemy
 - LSI: effective but slow
- We introduced 2 techniques to improve TM search by using query expansion
- We compared them with TM and LSI search:
 - we used a non-standard mid-size dataset
 - generally their performances are better than TM, but not homogeneous




What we have done

- Previous techniques:
 - TM: fast but prone to synonymy and polysemy
 - LSI: effective but slow
- We introduced 2 techniques to improve TM search by using query expansion
- We compared them with TM and LSI search:
 - we used a non-standard mid-size dataset
 - generally their performances are better than TM, but not homogeneous

What we want to do

- Further experiments on standard datasets
- Exploit user relevance feedback to discriminate relevant concepts in case of ambiguous queries
- Very big data structures: can we decrease spatial cost?

Bibliography I

-  H. Bast and D. Majumdar.
Why spectral retrieval works.
In Proceedings of ACM SIGIR, 2005.
-  S. Deerwester, S.T. Dumais, T.K. Landauer, G.W. Furnas,
and R.A. Harshman.
Indexing by latent semantic analysis.
J.Soc.Inform.Sci., 41(6):391–407, 1990.
-  C. Papadimitriou, P. Raghavan, H. Tamaki, and S. Vempala
Latent semantic indexing: A probabilistic analysis.
J. Comput. Systems Sciences, 61(2):217–235, 2000.

Bibliography II



Y. Qiu and H. Frei.

Improving the retrieval effectiveness by a similarity thesaurus.

Technical Report 225, ETH Zurich, 1994.

Thank you