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

Luigi Laura, Umberto Nanni, and Fabiano Sarracco

Dip. di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria, 113 - 00198 Roma Italy.
{laura,nanni,sarracco}@dis.uniroma1.it

Abstract. It is widely known that spectral techniques are very effective for document retrieval. Recently, a lot of effort has been spent by researchers to provide a formal mathematical explanation for this effectiveness [3]. Latent Semantic Indexing, in particular, is a text retrieval algorithm based on the spectral analysis of the occurrences of terms in text documents. Despite of its value in improving the quality of a text search, LSI has the drawback of an elevate response time, which makes it unsuitable for on-line search in large collections of documents (e.g., web search engines). In this paper we present two approaches aimed to combine the effectiveness of latent semantic analysis with the efficiency of text matching retrieval, through the technique of query expansion. We show that both approaches have relatively small computational cost and we provide experimental evidence of their ability to improve document retrieval.

1 Introduction

One of the most important tasks of an information retrieval system is to return, in response to a user query generally expressed as a set of terms, the subset of the managed documents which best matches the information necessity expressed by the user. Text matching is, beyond doubt, the most common technique used to accomplish this document retrieval task; the search is performed by looking for (and returning to user) the documents which contain the set, or a subset, of the terms included in the query. Two classical factors may negatively affect the quality of text matching search: *polysemy*, i.e., the same term may have different meanings (e.g., "web"), and *synonymy*, i.e., two distinct terms with the same meaning may be used interchangeably (e.g., "car" and "automobile"). In the former case, documents that are not relevant may be erroneously returned, while in the latter case, documents relevant to the user query may not be returned. To avoid these two and other similar problems, one would ideally like to represent documents (and queries) not by terms, but by the underlying (latent, hidden) concepts referred to by the terms. Unfortunately, as many works clearly point out, it is not possible to once-and-for-all define a universal terms-concepts

mapping structure, since this one heavily depends on the specific collection of documents we are dealing with.
Latent Semantic Indexing (LSI), introduced by Deerwester *et al.* in [5,6], is a technique to automatically compute from a documents collection a sort of semantic structure; this is achieved by applying a spectral decomposition technique, like the Singular Value Decomposition, on the data structure representing the occurrence of terms in documents. Documents (and queries) are represented and compared into this reduced "concept" space where, possibly, a document can be adjacent to (and thus relevant for) a query, even if the two do not have any term in common, but nevertheless share some underlying concept. LSI proved to be a very effective retrieval technique: it has been repeatedly reported that LSI outperforms classical text matching techniques, also addressing the problems of polysemy and synonymy[1,12]. This success has been empirically attributed to its ability to reduce noise, redundancy and ambiguity in the managed data structures. One of the still unsolved central questions on LSI is that the computational cost of the retrieval process strongly depends on the size of the collection. This makes LSI not appropriate for retrieving documents from huge collections, when we want to provide a fast online query answer.

Therefore we have on one side text matching, with fast response time but unsatisfactory filtering capabilities on the the document base, and on the other side LSI and similar techniques having powerful selection capabilities but high computational costs. In this paper we present two techniques that take advantage of the information provided by LSI data structures to enhance the quality of text matching search without increasing its response time. Both approaches process the user query by altering the set of terms included in it. We wish to remark that even if we present our approaches as a way to improve the effectiveness of text matching, the techniques introduced here, and more generally query expansion techniques, are fairly independent from the underlying retrieval algorithm used.
Our first technique, that we named *LS-Thesaurus*, is inspired by the work by Qiu and Frei [14]: they showed that the retrieval effectiveness of text matching can be improved by using a thesaurus, i.e., a structure indicating the similarity between each pair of terms. The idea is simple and natural: if terms in the query are relevant to the documents the user is looking for, these documents should contain also terms similar to the ones in the query. Consider the following example: suppose that the query "`football games`" is provided, and that this paper is included in the document collection. A text matching search would certainly include this document in the answer set but, obviously, this is not a document relevant to *football games*. However, if we expand the query by adding terms that are conceptually similar to the ones included in it, relevant documents are likely to be ranked higher. Our approach differs from the original work of Qiu and Frei because we make use of the data structures provided by the LSI technique, to generate the thesaurus.
LS-Filter, that is the second technique we present in this paper, can be intuitively described as a way to put in the query the more appropriate

terms for retrieving documents "conceptually related" to the query. To do this we project the query in the reduced concepts space generated by LSI; here we emphasize the largest components, that are the "important" concepts embedded in the query, and at the same time we set to zero the components with a small value, i.e., that are not relevant to the query. The modified vector is projected back in the terms space; our hope is that this new set of terms is better representative of the most important concepts in the query.
The rest of the paper is organized as follows: Section 2 provides the necessary background to understand our techniques, detailed respectively in Section 3 and Section 4. We present preliminary experimental results in Section 5, and final remarks are addressed in Section 6

2 Basic concepts and notation

In this section we define the preliminary concepts and techniques required to describe our main results. While doing this, we also fulfill the twofold task of fixing the notation used in the rest of the paper and presenting the relevant works in literature.

2.1 Term-document matrix and Text-Matching search

Consider a collection $D = \{d_1, \ldots, d_n\}$ of n text documents. Let $T = \{t_1, \ldots, t_m\}$ be the set of distinct index terms in D. The index terms are usually a subset of all the distinct terms occurring in the documents in D. Generally, this subset is composed only of words that are "relevant to retrieve the document" thus, for example, common words like articles and connectives are not considered as index terms. The *term-document* matrix of D is an $m \times n$ matrix A, representing a function of the occurrences of index terms in the documents in a compact way that can be efficiently managed and used by a text retrieval system. Matrix A has the following structure:

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

where element $a_{i,j}$ is generally a real number indicating the relative weight of term t_i in document d_j. In its simplest form, A is a binary matrix where:

$$a_{i,j} = \begin{cases} 1 & \text{if term } t_i \text{ occurs in document } d_j \\ 0 & \text{otherwise} \end{cases}$$

In literature a vast set of more refined term-weighting schemes are shown to provide, at least for particular data sets, better retrieval performances.

In the rather general *tf-idf* (term frequency-inverse document frequency) weighting scheme, for instance, the value of element $a_{i,j}$ is a function of two orthogonal factors:

- A *local factor* (or *term frequency factor*) $L(i,j)$, measuring the influence (or relevance) of term t_i in document d_j. A commonly used formula for $L(i,j)$ is:

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

  where $freq(i,j)$ is the frequency of term t_i in document d_j, i.e., the number of occurrences of t_i in d_j, divided by the total number of index terms in d_j.
- A *global factor* (or *inverse document frequency factor*) $G(i)$, whose aim is to de-amplify the relative weight of terms which are very frequently used in the collection. A commonly used formula for $G(i)$ is:

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

  where n_i is the number of documents containing term t_i.

See [15] for a more detailed description of the various term weighting schemes presented in literature.

While using a text retrieval system, the user submits a set of (possibly weighted) terms which, by his knowledge, best represent the information he is looking for. In our vectorial model, we can represent a user query as an m-dimensional vector $\boldsymbol{q} = (q_1, \ldots, q_m)$ (the *query vector*), where $q_i = w_i$ if term t_i is present in the query with weight w_i (or $q_i = 1$ if terms are not weighted), and $q_i = 0$ otherwise. In the following we assume $0 \leq q_i \leq 1$, for all $i \in [1, m]$.
In response to a user query, the system returns a score for each document in D, indicating its relevance to the query. Formally, the output of a search algorithm is an n-dimensional vector $\boldsymbol{r} = (r_1, \ldots, r_n)$ (the *rank vector*), where the value r_i measures the *relevance* of document d_i for the query expressed by $\boldsymbol{q}$. We assume that if $r_i > r_j$ then document d_i is more relevant than document d_j for the given query.

Let A be a term-document matrix representing a collection D of documents, and let $\boldsymbol{q}$ be a non zero query vector[1] given as input to the *text-matching search algorithm* (or simply TM). Algorithm TM returns as output the following rank vector:

$$\boldsymbol{r} = \boldsymbol{q} \cdot A$$

Note that A is a sparse (possibly binary) matrix and $\boldsymbol{q}$ is a sparse (possibly binary) vector. Therefore the matrix-vector product $\boldsymbol{q} \cdot A$ can be computed very efficiently. For instance, if, as commonly happens, the size of the query is limited to k terms, then Algorithm TM on a document base of n documents has cost $O(kn)$.

[1] Here we are assuming that at least one term of the query occurs in the collection

2.2 Query Expansion and Similarity Thesaurus

By *query expansion* or, more generally, *query reweighting*, we mean the process aimed to alter the weights, and possibly the terms, of a query. Usually, this process can be driven by the feedback provided by the user to the system (for instance, by selecting, among the retrieved documents, the ones that are considered more interesting). In this case the target is to refine a previously performed search. Indeed, a query can be reweighted also by exploiting a "knowledge", stored in the system, about terms usage in the particular collection of documents under exam. For instance, if in the collection of documents the term "automobile" is used often, then the system should add this term to a query containing only the term "car". In this paper we focus on this latter approach.
A commonly used method to have a statistic estimation of relationships among terms in a collection of documents, is to compute the term-term correlation matrix $A\,A^T$. If, for instance, we use the *tf-idf* weighting scheme, then the entries of $A\,A^T$ measure the co-occurence of terms in documents.
Qiu and Frei present in [14] an alternative approach to measure the term-term relation. Their idea is to compute the probability that a document is representive of a term. To do that, they propose the following weighting scheme:

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

where, as before, $freq(i,j)$ is the frequency of occurences of term t_i in document d_j and, for each term t_i, $\max_j freq(i,j)$ is the maximum frequency of term t_i over all the documents in the collection. Furthermore, let m_j the number of distinct terms in the document d_j, the ***inverse term frequency*** is defined as,

$$itf_j = \log \frac{m}{m_j} \tag{3}$$

In order to distinguish this particular term-document matrix from the ordinary one defined in Section 2.1, we denote it as $\overline{A}$.
A *similarity thesaurus* is a term-term correlation matrix S defined as:

$$S = \overline{A}\,\overline{A}^T.$$

A query vector $\boldsymbol{q}$ can be reweighted by using matrix S in the following way. First the vector $\boldsymbol{s} = \boldsymbol{q}S$ is computed. Then a threshold function $\xi(\boldsymbol{s}, x_r)$ is applied to $\boldsymbol{s}$. Assuming that $\boldsymbol{s} = (s_1, \ldots, s_m)$, function $\xi(\boldsymbol{s}, x_r)$ returns an m-dimensional vector $\boldsymbol{s}' = (s'_1, \ldots, s'_m)$ such that:

$$s'_i = \begin{cases} s_i & \text{if } s_i \text{ is among the } x_r \text{ largest (absolute) elements in } \boldsymbol{s} \\ 0 & \text{otherwise} \end{cases}$$

In other words, function ξ sets to zero all the elements of $\boldsymbol{s}$ which are not among the largest x_r elements, considering their absolute value. Note

that computing the function ξ requires time linear in the size of $\boldsymbol{s}$. The obtained vector $\boldsymbol{s}'$ is then normalized, by dividing all its elements by $|q| = \sum_{i=1}^{m} q_i$, thus obtaining a new vector $\boldsymbol{s}''$. Finally, the expanded query $\boldsymbol{q}'$ is computed as $\boldsymbol{q}' = \boldsymbol{q} + \boldsymbol{s}''$.
Query expansion techniques dates back to the work done by Maron and Kuhns in 1960 [9]. Automatic query expansion, despite the not encouraging results shown in the early work done by Lesk [8], Sparck Jones and Barber [16] and Minker *et al.* [10], where it seemed not an effective technique, has been revaluated after the researches made by Vorhees [18], by Crouch and Yang [4] and by Qiu and Frei [13].
The use of a thesaurus to improve the retrieval effectiveness has been introduced by Qiu and Frei in [14], where the authors use a global similarity thesaurus, and in [4] by Crouch and Yang. The difference in their approaches is on the type of the thesaurus: Qiu and Frei use a similarity thesaurus, i.e., a thesaurus based on the similarity between terms; the thesaurus proposed by Crouch and Young is statistical, i.e., is based on statistical co-occurences of the terms.

2.3 Latent Semantic Indexing

If we look at matrix A as a set of n distinct column vectors, we can interpret the elements of each column vector as the coordinates of the corresponding document in the m-dimensional subspace spanned by the terms in the collection. Intuitively, the more two documents (or a document and a query) are similar, the more the corresponding vectors are "close" to each other in the m-dimensional subspace. Polysemy and synonymy issues get their own interpretation in this context: along the same vector may lay two or more distinct meanings (polysemy), or, vice versa, the same concept may be spread along different vectors (synonymy).
The main idea behind *Latent Semantic Indexing* is to pre-compute a k-rank approximation of matrix A (with $k \ll m$), and to score documents according to their proximity to the given query vector in this low dimensional subspace. Experimentally, LSI turned out to be very effective in documents retrieval, overcoming, in most cases, the problems of polysemy and synonymy. A widely accepted intuitive explanation is that, more than terms, are the "latent semantic" concepts that should index the documents and the query.

As we said, LSI algorithm has a preprocessing offline phase. During this step the Singular Value Decomposition (or simply SVD) of A is computed. This is a way to rewrite A as a product of three matrices:

$$A = U\,\Sigma\,V^T$$

where:
U is a $m \times m$ orthonormal matrix [2], whose column vectors are the eigenvectors of $A\,A^T$;

[2] A square matrix M is orthonormal if is regular ($\det M \neq 0$) and if $M^T = M^{-1}$; so, for a orthonormal matrix M it holds $M^{-1} \cdot M = M^T \cdot M = I$.
Also $\det M = \pm 1$.

V is a $n \times n$ matrix whose column vectors are the eigenvectors of $A^T A$;
$\varSigma$ is a $m \times n$ matrix made as follows:

$$\varSigma = \begin{bmatrix} \varSigma_r & 0 \\ 0 & 0 \end{bmatrix}$$

where $\varSigma_r = diag(\sigma_1, \ldots, \sigma_r)$. The values $\sigma_1, \ldots, \sigma_r$ are called *singular values*, are decreasingly ordered, and correspond to the square roots of the eigenvalues of $A\,A^T$. Note that r is the rank of matrix A, and obviously $r \leq n$.

We get a k-rank approximation A_k of A, by considering only the largest k singular values and fixing the remaining $(r-k)$ singular values to zero. Therefore,

$$A_k = U_k\, \varSigma_k\, V_k^T \tag{4}$$

where:

- $U_k = U^{(1:m,1:k)}$ is the $m \times k$ matrix composed by the first k columns of U; [3]
- $V_k = V^{(1:n,1:k)}$ is the $n \times k$ matrix composed by the first k columns of V;
- $\varSigma_k = \varSigma^{(1:k,1:k)}$ is the diagonal $k \times k$ matrix whose elements are the k largest singular values.

Note that A_k has still size $m \times n$. A well known theorem by Eckart and Young (see [7]) states that, among all the $m \times n$ matrices of rank at most k, A_k is the one that minimizes the distance from A, measured by the Frobenius norm.

The i-th row vector of V_k defines the components of document d_i in the k-dimensional subspace spanned by the column vectors of U_k. Observe that $V_k = A^T\, U_k\, \varSigma_k^{-1}$. When a user query $\boldsymbol{q}$ is given as input to algorithm LSI, the projection of $\boldsymbol{q}$ in the k dimensional subspace is computed as: $\boldsymbol{q}_k = q\, U_k\, \varSigma_k^{-1}$. Then the score of each document d_i is obtained by measuring the distance between the i-th row vector of V_k and the query vector $\boldsymbol{q}_k$. One way to do this is to compute the cosine of the angle between the two vectors:

$$r_i = \cos\left(\boldsymbol{q}_k, V_k^{(i:i,1:k)}\right)$$

where $V_k^{(i:i,1:k)}$ is the i-th row of V_k. The computational cost of LSI is thus $O(kn)$. Note that, unlike TM, the matrices V_k, U_k and the vector $\boldsymbol{q}_k$ are dense and therefore their product cannot be efficiently computed. This computational cost, which heavily depends on the size of the collection of documents, makes LSI unsuitable for online applications dealing with huge collections of documents, like, for instance, web search engines.

Papadimitriou *et al.* present a formal probabilistic analysis of the LSI as a IR tool in [12]. Azar *et al.* [1], in a more general context from Papadimitriou *et al.*, justify from a theoretical point of view both the empirical success of LSI and its effectiveness against the problem of polysemy.

[3] In this paper we use the colon notation to define submatrices: by $A^{(i:j,k:l)}$ (with $i \leq j$ and $k \leq l$) we denote the submatrix of A having element $a_{i,k}$ as the upper left element and $a_{j,l}$ as the lower right element.

Algorithm 1 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 $\overline{A}$ from $D$;
5: $(U, \Lambda) \leftarrow EIGEN(\overline{A}\,\overline{A}^T)$;
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 2 Algorithm LS-Thesaurus Expand

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

3 LS-Thesaurus Algorithm

In this section the first of our two approaches is presented. The idea here is to generate a similarity thesaurus like the one described in Section 2.2. In this case, however, the similarity matrix S is computed starting from a low rank approximation of $\overline{A}$ (as in LSI). We can formally define our similarity matrix S_k in the following way:

$$S_k = \overline{A}_k\, \overline{A}_k^T = \overline{U}_k\, \overline{\Sigma}_k\, \overline{V}_k^T\, \overline{V}_k\, \overline{\Sigma}_k^T\, \overline{U}_k^T = \overline{U}_k\, \overline{\Sigma}_k^2\, \overline{U}_k^T \tag{5}$$

Note that $\overline{U}$ and the diagonal elements of $\overline{\Sigma}$ correspond respectively to the eigenvectors and the eigenvalues of matrix $\overline{A}\,\overline{A}^T$. Therefore, in the preprocessing step, we compute the eigenvalues and the eigenvectors of matrix $\overline{A}\,\overline{A}^T$ (through the function $EIGEN()$). We assume that the set of eigenvalues $\lambda_1, \ldots, \lambda_r$ is returned in the form of a diagonal matrix $\Lambda = diag(\lambda_1, \ldots, \lambda_r)$, while the eigenvectors are returned as the column vectors of a matrix U. The preprocessing step ends by returning the similarity matrix S_k computed as described in Equation (5). The query expansion is performed in the same way as in the original similarity thesaurus.

4 LS-Filter Algorithm

In this section we present our second query expansion technique, denoted as *LS-Filter*. Here we start from the following assumption. In the LSI algorithm documents and queries are projected (and compared) in

Algorithm 3 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: $(U, \Sigma, V) \leftarrow SVD(A)$;
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 4 Algorithm LS-Filter Expand

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

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. However, it may happen that, due to user obliviousness or to intrinsic properties of the collection, the set of terms in the user query is not the best suitable for a text matching search. What algorithm *LS-Filter* tries to do is to "guess" the concepts the user is indicating with its query and to find the most suitable terms for retrieving, by a text matching search, the most relevant documents for these concepts.
This is achieved by projecting the query vector $\boldsymbol{q}$ into the reduced ("concepts") space generated by LSI (through a precomputed matrix $P = \Sigma_k^{-1} U_k^T$). The k-dimensional vector $\boldsymbol{p}$ we obtain measures the relation between the query and each of the k (latent) concepts contained in the collection. The algorithm filters it by setting to zero the elements having small absolute values (by using function ξ). The filtered vector $\boldsymbol{p}'$ is then projected back in the terms space, thus obtaining a vector $\boldsymbol{p}''$ which provides the (weighted) terms that are the counterimage of the concepts is the query. Finallly, we apply again function ξ, thus leaving in the final expanded query $\boldsymbol{q}'$ only the x_t terms having the largest weights.

5 Experimental results

In this section we present the results of the experimental studies we accomplished so far. 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 as described in Section 2.3.

Dataset. Our data set are three books from O'Reilly (www.oreilly.com). They are publicly downloadable (in HTML version) from [11], and are an interesting (and difficult) dataset because they are quite specific (they all are related with computer science), and the correlation between them is high if compared to more heterogeneous collections of documents. We considered each HTML page as a different document, and indexed all the text of the page (excluding the tags). The overall number of documents is more than 3000, 1500 of which were from the first book, 1000 from the second book and the remaining 500 were from the last one. This collection is larger than the standard small datasets like CISI and MED [2] but its size allows a fast experimental setup, as opposite, for example, to the huge corpora provided for the TREC conferences [17].

Retrieval performance evaluation. To evaluate the effectiveness of these different IR techniques we used the standard precision versus recall diagrams at the eleven standard recall levels [2]. We briefly recall its definition.
Let $\boldsymbol{q}$ be a given user query. Let $R \subseteq D$ be the subset of documents which are relevant to $\boldsymbol{q}$ (usually determined by human experts), and let $|R|$ be the number of such documents. Moreover, we define A_h, for any integer $h \in [1, n]$, as the set of the h most relevant documents for $\boldsymbol{q}$ (we assume there are no ties), according to the rank vector returned by the system we are evaluating. Let $Ra_h = R \cap A_h$ be the intersection of the sets A_h and R. The recall and precision measures are defined as follows:
– $Recall_h$ is the ratio between the number of relevant documents in A_h and the total number of relevant documents:

$$Recall_h = \frac{|Ra_h|}{|R|}$$

– $Precision_h$ is the fraction of relevant document in A_h:

$$Precision_h = \frac{|Ra_h|}{|A_h|} = \frac{|Ra_h|}{h}$$

Note that the value of $Recall_h$ increases as the value of h increases (for $h = n$, $Recall_n = 1$).
We first compute the values of $Recall_h$ and $Precision_h$ for each $h = 1, \ldots, n$. Then we plot the precision-recall diagram by computing, through interpolation, the values of $Precision_h$, corresponding to the values of h for which $Recall_h = 0.0, 0.1, 0.2, \ldots 1$ (the standard recall levels).

Overview of the results. The precision-recall diagram for the tests we conducted, is plotted in Figure 1; we can observe that the techniques we propose performs between LSI and TM. It is interesting also to note

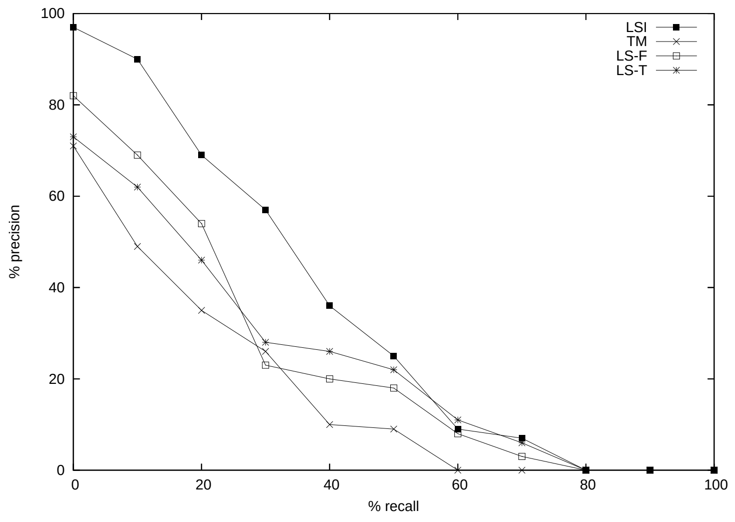

Fig. 1. Average precision at eleven standard recall levels

their relative performance: LS-T behaves better in the first half of the plot, and then is outperformed. To complete the picture, in Figure 2 is it shown the average of the times needed by each algorithm to respond a query, depending on the size of the collection. We notice that there are two different y-axis; the left one, whose range is from 0 to 60 seconds, measures the LSI while the right one, ranged 0 to 6 seconds, measures the other techniques (TM, LS-F and LS-Thaving roughly the same performances)[4]. The plot confirms at least one order of magnitude between LSI and the other techniques. These results indicate that the techniques we propose are a good trade-off between retrieval effectiveness and time performance.

Is it interesting to notice that, despite the (worst-case) running time of LS-T is obviously linear in the number of the terms in the collection (and definitely worse than LSI, linear in the number of documents), in practice the similarity thesaurus matrix is largely sparse (Qiu and Frei observed that usually around 3% of the entries are not zeroes). This means that every term can index its similar terms, and we can still provide a fast query answer. Consistently with the observation of Qiu and Frei our thesaurus has around 2,7% of non-zero entries.
We conclude by presenting, in Table 1, some examples of queries expanded by algorithm *LS-Filter*. We note that in some cases the system works as we would like to, for example when it adds the terms "bourne"

[4] We would like to point out that our IR system has been implemented to study and evaluate different techniques and therefore we focus only on the relative performances; in an optimized real IR system answer times can be significantly smaller.

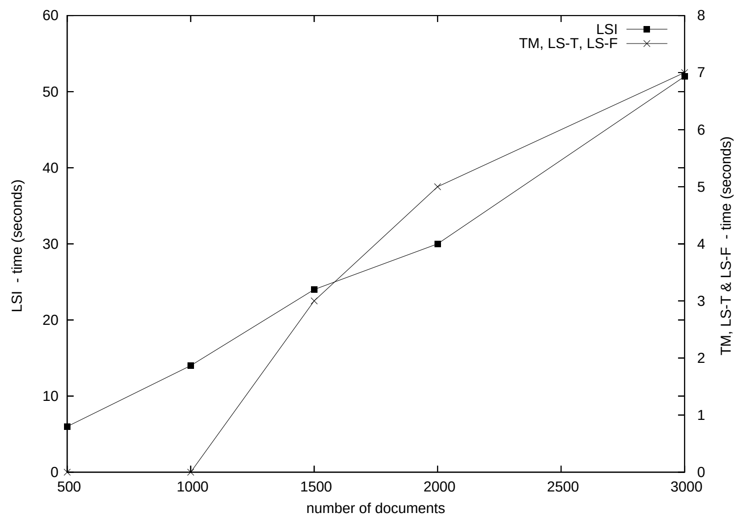

Fig. 2. Time comparison chart – left y-axis is for LSI, right y-axis for TM, LS-F and LS-T

and "prompt" to the query "shell logout"; sometimes, however, it adds misleading terms, like "ctrl" to the query "job control". We also see that the weight of the terms in the original query can be less than the one of the added terms; see for example the weight of term "control" in the query "job control".

6 Conclusions

In this paper we introduced two techniques, *LS-Thesaurus* and *LS-Filter*, that allow fast user query answer based on the LSI reduced space matrix. The preliminary experimental tests showed that our approaches perform well, and are a good trade-off between LSI and more simple methods like text matching. However, to evaluate the real effectiveness of both techniques we definitely need to perform more experiments on further data sets. We are working in this direction, and we are developing an open IR system that would allow to easily test different techniques.
We want to point out that the thesaurus generated according to *LS-Thesaurus* can be useful not only in the query expansion process, but also as a tool by itself. Moreover it can be used in an online style: the user can ask a query, see a list of similar terms and then he can decide to refine the query by adding some of these terms.
For the *LS-Filter*, we must remark its double behavior. In cases where the query terms correctly express the concept behind the query, this algorithm is able to outline and discard concepts of no interest, providing

java beans		job control		shell logout		zip file	
java	0.58136	job	0.42763	shell	0.68739	file	0.71548
bean	0.54392	background	0.12985	bourne	0.11818	util	0.16258
jdk	0.24954	echo	0.10754	login	0.09497	zip	0.15056
property	0.09444	number	0.09974	perl	0.09123	checksum	0.09776
nbsp	0.08534	list	0.07858	prompt	0.09851		
amp	0.07158	control	0.06929				
job	0.07119	ctrl	0.06882				
program	0.06206	object	0.06669				
value	0.05819	line	0.05827				
		filename	0.05827				

Table 1. Examples of query expanded by LS-Filter: user queries (first row) and the corresponding terms and weights.

more suited terms for the search. Sometimes (very often) query terms are still ambiguous and don't catch the real concept of interest. In these cases, we are not able to provide a correct retrieval; furthermore, the concept filtering can cut-off the concept of interest (with no high ranking), providing absolutely incorrect terms. The performances, in these cases, are vary bad, worse than simple text-matching. That is why in some cases the average graphics show for *LS-Filter* performances comparable to simple text-matching search.
This technique could be more effective if an appropriate user relevance feedback helps to discriminate relevant concepts in cases of ambiguous queries. In the system we are implementing, we want to provide the ability to select from the collection a set of terms with high discriminating power above concepts. In this way, every retrieved document is presented with all the discriminating terms it contains. Furthermore, everytime there is an ambiguity in the query among two or more concepts, the system is able to prompt the user for selection (/deselection) of the discriminating terms associated with ambiguous concepts. By selecting (/deselecting) one or more of these terms, or one or more of the retrieved documents, the user implicitly choices the concept of interest, allowing this way a more precise retrieval.

References

1. Y Azar, A. Fiat, A.R. Karlin, F. McSherry, and J. Saia. Spectral analysis of data. In *Proc. of STOC 2001*, 2001.
2. R. Baeza-Yates and R. Ribeiro-Neto. *Modern Information Retrieval.* Addison Wesley, 1999.
3. H. Bast and D. Majumdar. Why spectral retrieval works. In *Prooceedings of ACM SIGIR*, 2005.
4. C.J. Crouch and B. Yang. Experiments in automatical statistical thesaurus construction. In *Proc. of the 15th ACM SIGIR Conference on Research and Development in Information Retrieval*, 1992.

5. 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.
6. S.T. Dumais, G. Furnas, T.K. Landauer, and S. Deerwester. Using latent semantic analysis improve information retrieval. In *Proceedings of CHI'88: Conference Human Factors in Computing*, pages 281–285, 1988.
7. G. Golub and C. Reinsch. Handbook for Automatic Computation II, Linear Algebra. Springer-Verlag, New York, 1971.
8. M.E. Lesk. Word-word associations in document retrieval systems. *American Documentation*, 20(1):8–36, 1969.
9. M.E. Maron and J.L. Kuhns. On relevance, probabilistic indexing and information retrieval. *Association for Computing Machinery*, 1960.
10. J. Minker, G.A. Wilson, and B.H. Zimmerman. An evaluation of query expansion by the addition of clustered terms for a document retrieval system. *Information Storage and Retrieval*, 8(6):329–348, 1972.
11. The O'reilly dataset. http://www.dis.uniroma1.it/~laura/.
12. C. Papadimitriou, P. Raghavan, H. Tamaki, and S. Vempala. Latent semantic indexing: A probabilistic analysis. *J. Comput. Systems Sciences*, 61(2):217–235, 2000.
13. Y. Qiu and H. Frei. Concept based query expansion. In *Proc. of the 16th ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 160–169, 1993.
14. Y. Qiu and H. Frei. Improving the retrieval effectiveness by a similarity thesaurus. Technical Report 225, ETH Zurich, 1994.
15. G. Salton and C. Buckley. Term-weighting approaches in information retrieval. *Information Processing & Management*, 24(5):513–523, 1988.
16. K. Sparck Jones and E.O. Barber. What makes an automatic keyword classification effective. *Journal of the American Society for Information Sciences*, 22(3):166–175, 1971.
17. Text REtrieval Conference. http://trec.nist.gov/.
18. E.M. Vorhees. *The Effectiveness and Efficiency of Agglomerative Hierarchic Clustering in Document Retrieval*. PhD thesis, Cornell University, 1986.