On Scaling Latent Semantic Indexing for Large Peer-to-Peer Systems [*]

Chunqiang Tang
Dept. of Computer Science
University of Rochester
Rochester, NY 14627-0226
sarrmor@cs.rochester.edu

Sandhya Dwarkadas
Dept. of Computer Science
University of Rochester
Rochester, NY 14627-0226
sandhya@cs.rochester.edu

Zhichen Xu
Yahoo! Inc.
701 First Avenue
Sunnyvale, CA 94089
zhichen@yahoo-inc.com

ABSTRACT

The exponential growth of data demands scalable infrastructures capable of indexing and searching rich content such as text, music, and images. A promising direction is to combine information retrieval with peer-to-peer technology for scalability, fault-tolerance, and low administration cost. One pioneering work along this direction is pSearch [32, 33]. pSearch places documents onto a peer-to-peer overlay network according to semantic vectors produced using Latent Semantic Indexing (LSI). The search cost for a query is reduced since documents related to the query are likely to be co-located on a small number of nodes. Unfortunately, because of its reliance on LSI, pSearch also inherits the limitations of LSI. (1) When the corpus is large and heterogeneous, LSI's retrieval quality is inferior to methods such as Okapi. (2) The Singular Value Decomposition (SVD) used in LSI is unscalable in terms of both memory consumption and computation time.

This paper addresses the above limitations of LSI and makes the following contributions. (1) To reduce the cost of SVD, we reduce the size of its input matrix through document clustering and term selection. Our method retains the retrieval quality of LSI but is several orders of magnitude more efficient. (2) Through extensive experimentation, we found that proper normalization of semantic vectors for terms and documents improves recall by 76%. (3) To further improve retrieval quality, we use low-dimensional subvectors of semantic vectors to cluster documents in the overlay and then use Okapi to guide the search and document selection.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Clustering, Search Process

General Terms

Algorithms, Performance, Experimentation

Keywords

Latent Semantic Indexing, Dimensionality Reduction, Peer-to-Peer IR

[*]Work in this paper was done at the University of Rochester. Chunqiang and Sandhya were supported in part by NSF grants CCR-9988361, CCR-0219848, ECS-0225413, and EIA-0080124; by DARPA/ITO under AFRL contract F29601-00-K-0182; by the U.S. Dept. of Energy Office of Inertial Confinement Fusion under Cooperative Agreement No. DE-FC03-92SF19460; and by equipment or financial grants from Compaq, IBM, Intel, and Sun.

1. INTRODUCTION

According to a recent report [24], the unique information added each year exceeds 10^{18} bytes and is estimated to grow exponentially. This trend calls for equally scalable infrastructures capable of indexing and searching rich content such as text, music, and images. Meanwhile, Peer-to-Peer (P2P) systems [20] are gaining popularity quickly due to their scalability, fault-tolerance, and self-organizing nature, raising hope for building large-scale information retrieval (IR) systems at a low cost.

Building a P2P IR system, however, still remains particularly challenging. The fundamental problem that makes search in existing P2P systems (e.g., Gnutella) difficult is that, with respect to semantics, documents are randomly populated. Given a query, the system has to search a large number of nodes to find some relevant documents, rendering the system unscalable. To address this problem, we proposed pSearch [32, 33]. pSearch organizes nodes into an application-level overlay network and populates documents in the network according to document semantics derived from Latent Semantic Indexing (LSI) [3, 8]. The distance (e.g., routing hops) between two documents in the network is proportional to their dissimilarity in semantics. The search cost for a query is therefore reduced since documents related to the query are likely to be concentrated on a small number of nodes. Our initial results have shown the great promise of this approach [32]. pSearch can efficiently approximate a centralized implementation of LSI.

Unfortunately, our recent extensive experimentation with LSI reveals some limitations of LSI itself, which may also cripple pSearch's efficiency and efficacy because of its reliance on LSI. (1) When the corpus is large and heterogeneous, LSI's retrieval quality is inferior to methods such as Okapi [27]. (2) The Singular Value Decomposition (SVD) that LSI uses to derive low-dimensional representations (i.e., *semantic vectors*) of documents is not scalable in terms of both memory consumption and computation time.

In this paper, we propose techniques to address these limitations of LSI and show their use in the pSearch framework.

- To improve the efficiency of LSI, we propose an algorithm we call eLSI (efficient LSI) to reduce the size of the input matrix for SVD while retaining the matrix's important content. We partition documents into clusters and use the centroids of the clusters as "representative" documents. We further reduce the dimensionality of the centroid vectors by filtering out elements corresponding to low-weight terms. The resulting matrix, which has short centroid vectors as columns, is several orders of magnitude smaller than the original matrix. Finally we apply SVD to this matrix to derive the basis of the semantic space. Experiments show that eLSI retains the retrieval quality of LSI but is several orders of magnitude more efficient. It outperforms four major fast dimensionality reduction methods [6, 9, 15, 23] in retrieval quality.

- We conducted extensive experiments with LSI using a large corpus and found that proper normalization of semantic vectors for terms and documents improve recall by 76% compared with the standard LSI that strictly follows SVD.
- Without sufficient dimensions, LSI cannot accurately rank documents for large corpora and is therefore noticeably inferior to Okapi. Unlike works that use LSI to improve retrieval quality, we use LSI as an *implicit* document clustering method that can work with low-dimensional data. [1] We use low-dimensional subvectors of semantic vectors to *implicitly* cluster documents in an overlay, which helps reduce the search space. We then use Okapi to guide the search process and document selection.

It should be emphasized that our contributions are beyond their use in pSearch, since the problems we address are common to many other systems. (1) Deriving low-dimensional representation for high-dimensional data is a common theme for many fields. Existing methods such as Principal Component Analysis (PCA) and LSI are not scalable. eLSI is efficient and produces high-quality low-dimensional data. Therefore it can be used in many systems to replace PCA or LSI. (2) The proper configuration we found for LSI should be of general interest to the LSI community. (3) Existing LSI implementations compare the semantic vector of a query with that of every document. Dumais noticed the inefficiency of this method and commented that no known technique can effectively reduce the search space for high-dimensional data [10]. The fundamentals of our techniques, despite the fact that they were originally developed for P2P systems, can also be applied to centralized systems to reduce the search space.

This paper addresses the challenge of using LSI in pSearch. One can also build a P2P IR system around our document clustering idea without using LSI, by partitioning documents into clusters and assigning them to different nodes. Given a query, it searches only nodes whose centroids are the closest to the query. Since nearest neighbor search in a high-dimensional space is prohibitive, this approach cannot employ a distributed search strategy as pSearch does. In a naive implementation, each node would need to know the IP and centroid of all other nodes. However, it is important for nodes in a dynamic P2P system to maintain only a small amount of global information to be scalable. It is a subject of future work to pursue this approach and address this challenges.

The remainder of the paper is organized as follows. Section 2 gives an overview of the pSearch system. Sections 3 and 4 describe and evaluate techniques to improve LSI's retrieval quality and efficiency, respectively. Section 5 puts all these techniques together and evaluates the complete pSearch system. Related work is discussed in Section 6. Section 7 concludes the paper.

2. SYSTEM OVERVIEW

To set the stage for our discussion, we first present an overview of the pSearch system (see [32] for details). In pSearch, a large number of nodes are organized into an application-level overlay network to offer IR service. Nodes in the overlay collectively form a pSearch *Engine*. Inside the Engine, nodes have completely homogeneous functions. A client intending to use pSearch connects to any Engine node to publish document indices or submit queries.

[1]Working with low-dimensional data is essential. Due to the *curse of dimensionality* [35], for a high-dimensional space implemented in a decentralized environment, the system has to search a large number of nodes to answer queries.

Figure 1 shows an example of how the system works. Node A publishes a document to node B inside the Engine. B builds the index for the document and routes the index in the overlay. The index is finally stored on node F based on its semantics. When a query is submitted to node E, the query is routed to node C based on the semantics of the query. C then takes the responsibility for finding relevant documents and returning them to E. In this example, C may return the index published by A and stored on F.

pSearch uses a CAN [25] to organize Engine nodes into an overlay and uses an extension of LSI to answer queries. We call this algorithm pLSI. In the following, we first give some background and then present the pLSI algorithm.

2.1 Vector Space Model (VSM)

In VSM [28], a term-document matrix $A = (a_{ij}) \in \mathcal{R}^{t \times d}$ is formed to represent a collection of d documents containing words from a vocabulary of t terms. Each column vector a_j ($1 \le j \le d$) corresponds to a document j. Weight a_{ij} represents the importance of term i in document j. The weights are usually computed from variants of TFIDF [3]. For instance, the *ltc* [7] term weighting scheme computes a_{ij} as follows,

$$b_{ij} = [\log(f_{ij}) + 1] \cdot \log(\frac{d}{D_i}) \quad (1)$$

$$a_{ij} = \frac{b_{ij}}{\sqrt{\sum_{x=1}^{t} b_{xj}^2}} \quad (2)$$

where f_{ij} is the frequency of term i in document j and D_i is the number of documents that contain term i. Normalization in Equation 2 ensures that document vector a_j is of unit length. Queries are represented as vectors in a similar fashion. The similarity between two vectors is measured as their inner product. When vectors are normalized (as they are in ltc), the inner product is the same as the cosine of the angle between the vectors.

2.2 Latent Semantic Indexing (LSI)

Literal matching schemes suffer from synonyms and noise in documents. LSI overcomes these problems by using statistically derived concepts instead of terms for retrieval. It uses truncated Singular Value Decomposition (SVD) [12] to transform a high-dimensional document vector into a lower-dimensional *semantic vector*, by projecting the former into a semantic subspace.

Suppose the rank of the term-document matrix A is r. SVD decomposes A into the product of three matrices,

$$A = U \Sigma V^T \quad (3)$$

where $U = (u_1, \ldots, u_r) \in \mathcal{R}^{t \times r}$, $\Sigma = \text{diag}(\sigma_1, \ldots, \sigma_r) \in \mathcal{R}^{r \times r}$, and $V = (v_1, \ldots, v_r) \in \mathcal{R}^{d \times r}$. V^T is the transpose of V. σ_i's are A's singular values, $\sigma_1 \ge \sigma_2 \ge \ldots \ge \sigma_r$. U and V are column-orthonormal. LSI approximates A with a rank-k matrix

$$A_k = U_k \Sigma_k V_k^T \quad (4)$$

by omitting all but the k largest singular values, where $U_k = (u_1, \ldots, u_k)$, $\Sigma_k = \text{diag}(\sigma_1, \ldots, \sigma_k)$, $V_k = (v_1, \ldots, v_k)$.

Row i of $U_k \in \mathcal{R}^{t \times k}$ is the representation of term i in the k-dimensional semantic space. A document (or query) vector $q \in \mathcal{R}^{t \times 1}$ can be folded into the k-dimensional semantic space using Equation 5 or 6 [23]. The difference is whether to scale the vector by the inverse of the singular values. Similar to VSM, the similarity between semantic vectors is measured as their inner product.

$$\hat{q} = U_k^T q \quad (5)$$

$$\hat{q} = \Sigma_k^{-1} U_k^T q \quad (6)$$

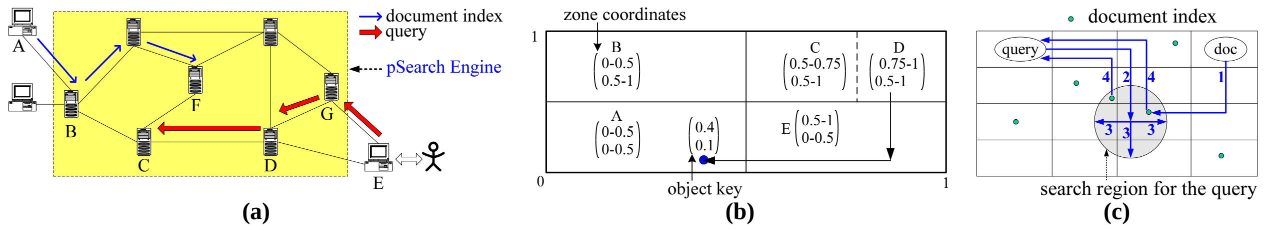


Figure 1: (a) Overview of the pSearch system. (b) A 2-dimensional CAN. (c) pLSI in a 2-dimensional CAN.

2.3 Content-Addressable Network (CAN)

Recent P2P overlay networks such as CAN [25] offer an administration-free and fault-tolerant storage space. The basic functionality these systems provide is a distributed hash table (DHT) that maps "keys" to "objects". CAN partitions a d-dimensional Cartesian space into *zones* and assigns each zone to a node. Two nodes are routing neighbors in the overlay if their zones overlap in all but one dimension along which they abut each other. An object key is a point in the Cartesian space and the object is stored at the node whose zone contains the point. Locating an object is reduced to routing to the node that hosts the object.

An example CAN is shown in Figure 1(b). There are five nodes A-E in the overlay. Each node owns a zone in the Cartesian space. Initially C owns the entire zone at the upper-right corner. When D joins, the zone owned by C splits and part of the zone is given to D. When D wishes to retrieve the object with key $(0.4, 0.1)$, it sends the request to E and E forwards the request to A.

2.4 The pLSI Algorithm

Both LSI and CAN employ a Cartesian space. The pLSI algorithm splices them together to build pSearch. It sets the dimensionality of a CAN to be equal to that of LSI's semantic space (k). The index for a document is stored in the CAN using its semantic vector as the key. The effect is that indices stored close in the overlay are also close in semantics.

Figure 1(c) illustrates the basic steps of pLSI.

1. When receiving a new document, the Engine node derives its semantic vector using LSI and uses the semantic vector as the key to store its index in the CAN.
2. When receiving a query, the Engine node derives its semantic vector and routes the query in the CAN using the semantic vector as the key.
3. Upon reaching the destination, the query is flooded to nodes within a small radius r.
4. Nodes that receive the query do a local search and return references of the best matching documents to the query initiator.

Since indices of documents similar to the query (above a certain threshold) can be stored only within this radius r and we do an exhaustive search within this area, in theory, pLSI can achieve the same precision as LSI. Ideally, the search region should be small such that only a small number of nodes are involved in a search.

In practice, we do not know the boundary of the search region in advance. The search starts from the node whose zone contains the query point and gradually explores its neighboring nodes, guided by our *content-directed search* algorithm [32]. Each node samples content stored on its neighbors and uses the samples to decide which node to search next. It prefers searching nodes whose indices, judging from the samples, have high similarity to the query. The search is terminated when no better document is found on the most recently searched T nodes, where T is a tunable threshold.

LSI uses a k=50∼350 dimensional space for small corpora. Due to the *curse of dimensionality* [35], the size of the search region in Figure 1(c) grows quickly as the dimensionality of the space increases, meaning pSearch needs to search a large number of nodes to answer queries. Moreover, there is a dimensionality mismatch between CAN and LSI. The "real" dimensionality of a CAN cannot be higher than $l = O(\log(n))$, where n is the number of nodes in the system, $l \ll k$. Our *rolling-index* algorithm [32] addresses both problems. Intuitively, it partitions a k-dimensional semantic vector into multiple disjoint l-dimensional subvectors, and uses the subvectors as DHT keys to guide index placement and query routing. For instance, for a k-dimensional vector $(v_1, v_2, \cdots, v_k)$, $(v_1, v_2, \cdots, v_l)$ is the first subvector, $(v_{l+1}, v_{l+2}, \cdots, v_{2l})$ is the second one, and so forth. Given a document, we store its index at p places in the CAN using its first p subvectors as DHT keys. For convenience, we simply say subvectors with the same starting offset belong to the same *plane* and l is the dimensionality of each plane. Conceptually, the index for a document is stored on p different planes. During a search, we execute the pLSI algorithm in Figure 1(c) p times. Each time it uses a different subvector of the query to guide the search on a different plane. The final search results are a combination of the results from different planes. Our previous evaluations [32] show that pLSI can approximate LSI using a small number of planes, typically $p = 4$.

Generally, two similar subvectors cannot ensure their full vectors are also similar, but we find that this probability is significantly higher for semantic vectors than for random vectors, because of the significance of the low-dimensional elements of semantic vectors and the correlation among the elements. We demonstrate this through experiments with the TREC corpus (see Section 3.1 for details of our experiments). We first retrieve 15 documents for each TREC 7&8 query based on the similarity of the 300-dimensional semantic vectors. The results form set A. On each plane, we then retrieve $e \cdot 15$ documents for a query solely based on the similarity of the l-dimensional subvectors. Here, e is a constant multiplication factor. The results for the first four planes form set B. The "sv" series in Figure 2 are the average *accuracy* of set B with respect to set A, where accuracy $= \frac{|A \cap B|}{|A|}$. When $l = 25$ and $e = 128$, B is only 1.3% of the entire corpus, but it already covers 90% of the documents in set A. We also conduct the same experiment using

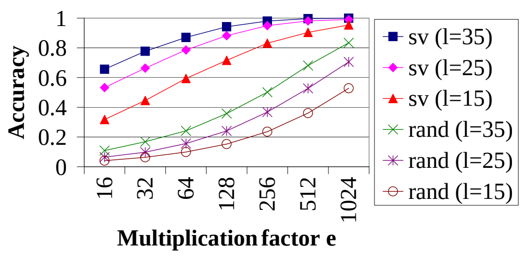


Figure 2: Using subvectors to search nearest neighbors.

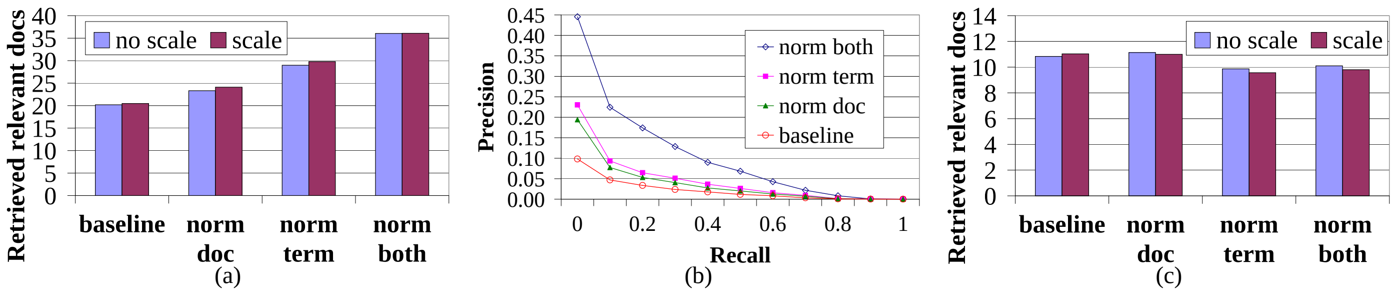


Figure 3: Comparison of different configurations for LSI. (a) Retrieved relevant documents for TREC. (b) Precision-recall for TREC. (c) Retrieved relevant documents for Medlars.

random vectors as documents and queries, and report the results as the "rand" series in Figure 2. This figure shows that, to some extent, similar semantic subvectors imply similar full semantic vectors, thanks to the features of semantic vectors.

A technique to balance index distribution across nodes is described in [32]. To derive the semantic vector for a document, Engine nodes need some global data, e.g., IDF and the basis of the semantic space (U_k). See technical report [33] for the method that aggregates and distributes these global data through a tree.

3. IMPROVING RETRIEVAL QUALITY

When the corpus is large and heterogeneous, LSI's retrieval quality is inferior to methods such as Okapi [27]. In this section, we describe techniques to improve LSI's retrieval quality.

3.1 Choosing a Proper LSI Configuration

Although the use of SVD is common among LSI implementations, we have seen proposals [4, 5, 8, 14] that differ in the manner in which the output of SVD is used, depending on

1. term normalization—whether to normalize rows of U_k (semantic vectors for terms) to unit length before using them in Equation 5 or 6 to project document or query vectors;
2. document normalization—whether to normalize the projected vectors ($\hat{q}$ in Equation 5 or 6) to unit length before using inner product to compute the similarity (i.e., the choice of using inner product or cosine as the similarity metric); and
3. the choice of using Equation 5 or 6 to project vectors.

There are a total of eight different variants of LSI depending on these choices. In analysis [23], LSI is usually treated as a process that uses a low-rank matrix to approximate a high-rank matrix while introducing the smallest error. The "standard" LSI that follows from this analysis should do neither term normalization nor document normalization and use Equation 6 to project vectors. Since the "standard" LSI is most widely used, we refer to it as the "baseline". Some systems in the literature have also used the "non-standard" variants. To the best of our knowledge, no study systematically evaluated these fundamental choices for LSI. Below we will show these choices dramatically affect LSI's performance.

To evaluate these choices, we extended the SMART system [7] with an LSI implementation, using SVDPACK [30] to compute SVD of large sparse matrices. The SMART stopword list and stemmer are used as is. The corpus we use is the disk 4 and 5 from TREC [34], excluding the Congressional Record. It consists of 528,543 documents with a total size of 2GB. Since real queries are usually short, we use the title field of topics 351-450 as queries. A query on average contains 2.4 terms and has 94 relevant documents.

We experimented with various term weighting schemes to generate the input term-document matrix for SVD, including Okapi [27], pivoted normalization [29], and those built in SMART. Although Okapi and pivoted normalization have good performance when used standalone, using them as a pre-processing step for LSI does not improve performance. The reason is that they use document length as one important factor in weighting, but it is hard to assign a length to short queries that can work with the equations in Okapi or pivoted normalization. We found that the ltc term weighting in Equations 1 and 2 works well with LSI for several corpora. We use ltc to generate the input term-document matrix for SVD.

Due to memory limitations, we select only 15% of the TREC corpus to construct a 83,098-term by 79,316-document matrix as the input for SVD, which projects vectors into a 300-dimensional space. The SVD computation consumes 1.7GB memory and takes 57 minutes to complete on a 2GHz Pentium 4 machine. We use LSI to retrieve 1,000 documents for each query and report the average number of retrieved relevant documents for a query in Figure 3(a). Figure 3(b) plots the precision-recall curve. [2] "no scale" uses Equation 5 to project vectors whereas "scale" uses Equation 6. "norm term" normalizes each row of U_k. "norm doc" normalizes semantic vector $\hat{q}$ in Equation 5 or 6 before computing similarity. "norm both" does both normalizations.

Normalizing semantic vectors of terms or documents significantly improves precision and recall. Combined together, they return 76% more relevant documents compared with the baseline that does no normalization. (In pSearch, we are more interested in LSI's recall since we use Okapi to rerank documents afterward. See Section 3.2.) The baseline, unfortunately, is widely used in most LSI implementations since it is what directly follows from SVD.

As pointed out in [14], normalizing semantic vectors for terms improves performance by emphasizing rare terms. Despite the compensation from the IDF component in Equation 1, rare terms tend to have a small norm after truncated SVD because their semantics are usually captured by high-dimensional elements that are truncated away. Consequently, rare terms contribute little to the final similarity score to differentiate documents. The benefit of normalizing document vectors again after SVD (the first normalization in Equation 2 is before SVD) corroborates the long-lasting belief that cosine is a robust measure for similarity.

We also conduct the same experiment with the Medlars corpus (available in the SMART package [7]), which has 1,033 documents and 30 queries. Documents and queries are projected into a 50-

[2] Precision is defined as the number of retrieved relevant documents divided by the number of retrieved documents. Recall is defined as the number of retrieved relevant documents divided by the total number of relevant documents in the corpus. A precision-recall curve shows the precision at a given recall level.

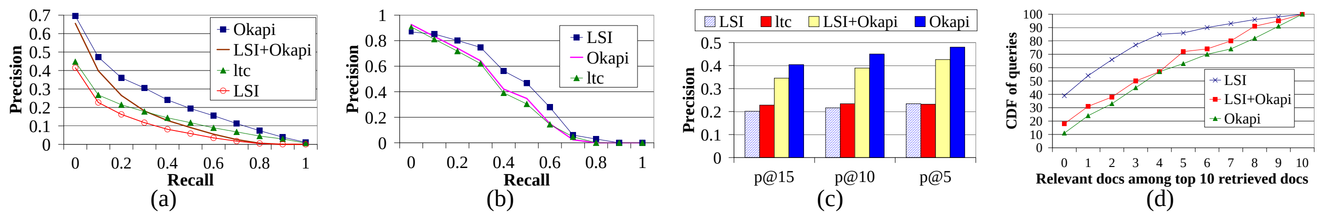


Figure 4: Comparison of different retrieval algorithms. (a) Precision-recall for TREC. (b) Precision-recall for Medlars. (c) High-end precision for TREC. (d) Cumulative distribution of queries as a function of retrieved relevant documents (using TREC).

dimensional semantic space. We retrieve 15 documents for each query and report the average number of retrieved relevant documents in Figure 3(c). Unlike that for TREC, LSI's performance for Medlars varies marginally with different configurations. We conjecture that the performance difference between Figure 3(a) and (c) is because a 50-dimensional space is sufficient for the small, homogeneous Medlars corpus whereas a 300-dimensional space is insufficient for the large, heterogeneous TREC corpus. Another experiment corroborates this conjecture. When using only an insufficient 15-dimensional space for Medlars, "norm both" outperforms the baseline by 30%, which is consistent with the trend for TREC. This experiment demonstrates the importance of using large, heterogeneous corpora in evaluations.

In summary, normalization is beneficial if the dimensions of the semantic space are insufficient in capturing the fine structure of the corpus, which is true for most large corpora. We choose "norm both" with Equation 5 as the configuration for LSI. There is no performance difference between using Equation 5 and 6. We opt for Equation 5 since it directly follows from the analysis [23].

3.2 Combining LSI with Okapi

Okapi [27] consistently achieved the best performance in TREC's ad hoc track. Figure 4(a) and (b) compares the precision-recall of several retrieval algorithms [3], using TREC and Medlars, respectively. (LSI+Okapi is our method to be described later.) LSI performs well for the small Medlars corpus, which is consistent with the results in previous work [8]. For the much larger TREC corpus, however, Okapi performs dramatically better than LSI. LSI's poor performance with large, heterogeneous corpora has also been reported elsewhere [14]. This experiment, again, emphasizes the importance of using large corpora in evaluations.

There are several reasons for LSI's inferior performance. First, LSI does not explicitly exploit document length in ranking, which has been shown to be important by Okapi and pivoted normalization [29]. There is no simple solution for this. Our experience shows that simply using Okapi or pivoted normalization to generate the input term-document matrix for SVD leads to even worse performance because of their asymmetric weights for document terms and query terms. Second, a 300-dimensional semantic space is insufficient for TREC. LSI's performance can be improved by increasing dimensionality (see results in Figure 5(a)), but this will increase the cost of SVD. More importantly, in a decentralized environment, pLSI only exploits information in low-dimensional subvectors to guide index placement and query routing (see Section 2.4). Increasing dimensionality is therefore not helpful for us.

SVD sorts elements in semantic vectors by decreasing importance. The low dimensions of the semantic space capture the major structure of the corpus, but it still needs the fine structure captured by the high dimensions to rank documents. In other words, regardless of the optimal dimensionality for LSI to achieve the best retrieval quality, low-dimensional subvectors can approximately cluster related documents in the P2P network. Without sufficient dimensions, LSI simply cannot rank documents properly.

Based on this observation, we make two important modifications to pLSI. First, on a searched node, we use Okapi instead of LSI to select documents. Second, we use Okapi to guide the exploration of the search region in Figure 1(c). In content-directed search, we use Okapi instead of LSI to compute the similarity between the sampled documents and the query, which is used to decide which node to search next. We call this method "LSI+Okapi".

We first evaluate if LSI+Okapi can work in a centralized implementation. Figure 4(c) compares the high-end precision (i.e., the precision when retrieving a small number of documents) for different methods. $P@i$ is the precision when retrieving i documents for a query. The configuration for LSI+Okapi is as follows. We use a 4-plane pLSI. Each plane is of 25 dimensions. That is, pLSI uses only the first 100 dimensions of the semantic space. Each plane retrieves 1,000 documents for a query based on subvectors on that plane. [4] Four planes in total return 4,000 documents. Finally, we use Okapi to rank the returned 4,000 documents. Figure 4(c) shows that, with proper ranking, the top documents retrieved by low-dimensional subvectors are almost as good as Okapi.

Figure 4(d) plots the cumulative distribution of the 100 queries as a function of the returned relevant documents when retrieving 10 documents. LSI, LSI+Okapi, and Okapi find no relevant document for 39, 18, and 11 queries, respectively. The distribution of the retrieval quality of LSI+Okapi closely follows that of Okapi.

The precision-recall for LSI+Okapi is reported in Figure 4(a). The high-end precision of LSI+Okapi approaches that of Okapi, but the low-end precision (i.e., the precision when retrieving a large number of documents) still lags behind. The low-end precision can be improved by allowing each plane to return more candidate documents for Okapi to rank, but this would increase the search cost. Currently our focus is on high-end precision since a significant percentage of users only view the top search results. We leave improving low-end precision as a subject of future work.

4. IMPROVING THE EFFICIENCY OF LSI

In Section 3, we described techniques to improve the retrieval quality of pLSI to approach that of Okapi. In this section, we address another problem that limits LSI's scalability—the high computation cost associated with SVD.

Traditionally, LSI uses a term-document matrix as the input for SVD to compute the basis of the semantic space. For a sparse matrix $A \in \mathcal{R}^{t \times d}$ with about c nonzero elements per column, the

[3]Although automatic query expansion can improve performance (e.g., boosting Okapi's average precision from 0.221 to 0.273), it is not used in any experiment in this paper, because we want to study the effect of dimensionality reduction independently.

[4]In a decentralized implementation, each plane does not actually return all 1,000 documents since content-directed search [32] automatically avoids searching documents with low similarity score.

time complexity of SVD is $O(t \cdot d \cdot c)$ [22]. We propose the following to reduce this cost. We partition documents into clusters and use the centroids of the clusters as "representative" documents. We further reduce the dimensionality of the centroid vectors by filtering out elements corresponding to low-weight terms. The resulting matrix, which has short centroid vectors as columns, is several orders of magnitude smaller than the original term-document matrix. Finally, we apply SVD to this matrix to derive the basis of the semantic space. We call this algorithm eLSI (efficient LSI). eLSI reduces the cost for SVD but introduces the extra clustering step. We believe clustering algorithms are much more scalable than SVD.

4.1 The eLSI Algorithm

Our eLSI algorithm efficiently derives good low-dimensional representation for documents. It consists of the following steps.

1. Partition documents into clusters and compute a centroid for each cluster.
2. Reduce the dimensionality of the centroids by keeping only elements whose aggregate weight across centroids is sufficiently large.
3. Project the dimensionality-reduced centroids into a k-dimensional semantic space using SVD.
4. Project terms into the semantic space according to the usage of terms in the centroids and the k-dimensional representation of centroids.
5. Finally, project documents into the semantic space according to the usage of terms in the documents and the k-dimensional representation of terms.

Intuitively, we use document clustering and term selection to come up with a small matrix that captures important content of the original term-document matrix. We then apply SVD to this small matrix to derive low-dimensional representation for the centroids, followed by a chain of actions, centroids $\rightarrow$ terms $\rightarrow$ documents, to derive low-dimensional representation for terms and documents.

Each column of the original term-document matrix corresponds to a document. One natural way to reduce the number of columns is to replace multiple columns corresponding to a cluster of similar documents with a single vector that represents the centroid of this cluster. Centroids are landmark structures in the document space. If we find a good projection that maps centroids to a low-dimensional semantic space while retaining the distance among them, it is likely that this projection also keeps the distance among documents. We use a hierarchical version of *spherical k-means* [9] to cluster documents. Details are omitted due to space limitation.

Denote the centroid matrix obtained through clustering as

$$C = [c_1\ c_2\ \cdots c_s] \in \mathcal{R}^{t \times s} \tag{7}$$

where s is the number of document clusters, t is the number of terms, and c_j is the centroid vector for cluster j. The centroid $\hat{v}$ of a set of vectors v_i $(1 \leq i \leq n)$ is defined as $\hat{v} = \frac{1}{n}\sum_{i=1}^{n} v_i$. Element c_{ij} indicates the importance of term i in centroid j. The aggregate weight of a term i across centroids is $w_i = \sum_{j=1}^{s} c_{ij}$. We select a subset of e rows from matrix C to construct a row-reduced matrix $\tilde{C} \in \mathcal{R}^{e \times s}$. The x-th row of C is kept in $\tilde{C}$ if term x appears in more than one centroid and is among the top e terms with the largest aggregate weight. The rationale behind term selection is that terms with big aggregate weight are representative in expressing the relationship among centroids.

Similar to the use of document clustering to reduce the columns of the term-document matrix, one can use term clustering to reduce the rows of matrix C. We prefer term selection since it is more efficient and works well in practice. Alternatively, one can also use Random Projection [11] to reduce the dimensionality of the centroid vectors. See Section 4.2 for more details.

After document clustering and term selection, matrix $\tilde{C}$ is several orders of magnitude smaller than the original term-document matrix. For the TREC corpus, the complete term-document matrix has 408,653 rows and 528,155 columns. The matrix $\tilde{C}$ we use for the TREC corpus, on the other hand, has less than 2,000 rows and 2,000 columns. We apply SVD to $\tilde{C}$, computing components corresponding to $\tilde{C}$'s k largest singular values (see Equation 8 and 9). This can be done efficiently because of the limited size of $\tilde{C}$.

$$\tilde{C} = U \Sigma V^T \tag{8}$$
$$\tilde{C}_k = U_k \Sigma_k V_k^T \tag{9}$$

$V_k \in \mathcal{R}^{s \times k}$ is the representation of the centroids in the k-dimensional semantic space. Equation 10 projects terms into the semantic space using V_k. Recall that each row of C corresponds to a term.

$$B = C V_k \in \mathcal{R}^{t \times k}. \tag{10}$$

As discussed in Section 3.1, we normalize each row of B to unit length to emphasize rare terms, resulting in a new matrix $\tilde{B}$. Finally, Equation 11 and 12 project a document (or query) vector q into the semantic space and normalize it to unit length.

$$\bar{q} = \tilde{B}^T q \tag{11}$$
$$\hat{q} = \frac{\bar{q}}{||\bar{q}||_2} \quad \in \mathcal{R}^{k \times 1} \tag{12}$$

4.2 Other Dimensionality Reduction Methods

In this section, we summarize four fast dimensionality reduction methods. We will compare them with eLSI in Section 4.3.

The first algorithm, Random Projection (RP) [11], projects a t-dimensional document (or query) vector q into a k-dimensional subspace using a random matrix $P \in \mathcal{R}^{t \times k}$ whose columns have unit length (see Equation 13). Here t is the number of terms and k is the dimensionality of the target semantic space, $k \ll t$. Despite its simplicity, previous work has shown that RP was reasonably accurate in reducing dimensionality for text data [6].

$$\bar{q} = P^T q \tag{13}$$

The first step of all other algorithms partitions documents into k clusters, where k is the dimensionality of the target semantic space. (Note that eLSI partitions documents into s clusters, $s \gg k$.) Denote $G = [g_1\ g_2\ \cdots g_k] \in \mathcal{R}^{t \times k}$ as the centroid matrix, where g_j is the centroid vector for cluster j.

The second algorithm, Concept Indexing (CI), is introduced by Karypis and Han [15]. Given a vector q, CI uses Equation 14 to project it into the subspace spanned by the k centroids.

$$\bar{q} = G^T q \tag{14}$$

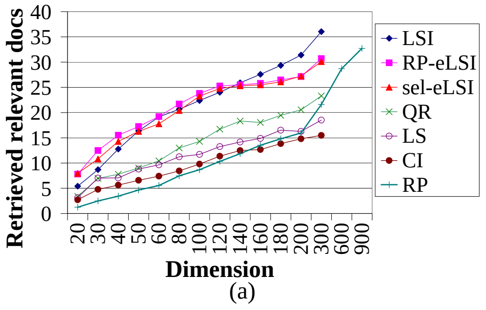
The third algorithm [9] solves the least-squares problem in Equation 15 to derive the k-dimensional representation for vector q. We will refer to this algorithm as the LS (least-squares) algorithm.

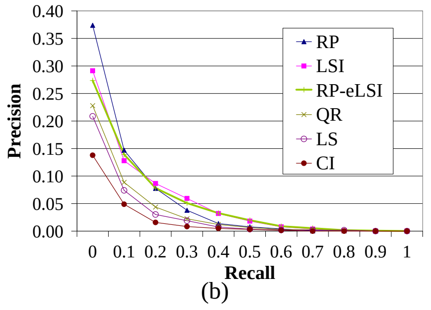
$$\bar{q} = \arg_{\bar{q}}\ \min ||G\bar{q} - q||_2 \tag{15}$$

The fourth algorithm [23] is based on QR decomposition. The QR decomposition of matrix $G \in \mathcal{R}^{t \times k}$ gives an orthogonal matrix $Q \in \mathcal{R}^{t \times t}$ and an upper triangular matrix $R \in \mathcal{R}^{k \times k}$ such that

$$G = Q \begin{pmatrix} R \\ 0 \end{pmatrix} = (Q_k\ Q_r) \begin{pmatrix} R \\ 0 \end{pmatrix} = Q_k R \tag{16}$$

where $Q_k \in \mathcal{R}^{t \times k}$ and $Q_r \in \mathcal{R}^{t \times (t-k)}$. The k-dimensional representation of vector q is computed using Equation 17. We will refer





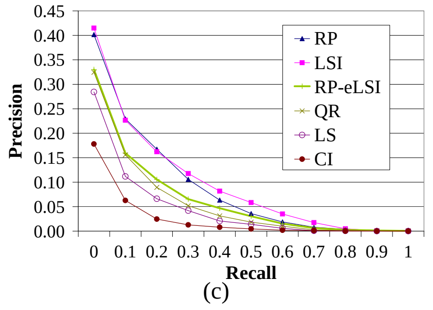


Figure 5: Comparison of dimensionality reduction methods. (a) Retrieved relevant documents. (b) Precision-recall in a 140-dimensional space. (c) Precision-recall in a 300-dimensional space.

to this algorithm as the QR algorithm.

$$\bar{q} = Q_k^T q \tag{17}$$

As discussed in Section 3, normalization improves retrieval quality. We use Equation 12 to normalize vector $\bar{q}$ in Equation 13, 14, 15, and 17 to unit length.

The eLSI algorithm described in Section 4.1 uses term selection to reduce the dimensionality of the centroid vectors before applying SVD. Alternatively, one can use random projection with eLSI to reduce the dimensionality of the centroid vectors,

$$\tilde{C} = F^T C \tag{18}$$

where $C \in \mathcal{R}^{t \times s}$ is the centroid matrix (from Equation 7), $\tilde{C} \in \mathcal{R}^{e \times s}$ is the reduced centroid matrix, and $F \in \mathcal{R}^{t \times e}$ is a random matrix whose columns have unit length. After obtaining $\tilde{C}$, other steps of the eLSI algorithm are used without change. We refer to this version of eLSI as "RP-eLSI" (using RP with eLSI) and refer to the original eLSI algorithm as "sel-eLSI" (term selection eLSI). We will use "eLSI" to generally refer to "sel-eLSI" and "RP-eLSI".

In total, we have seven different dimensionality reduction algorithms to compare—RP-eLSI, sel-eLSI, RP, CI, LS, QR, and LSI, where "LSI" is the traditional LSI algorithm that directly applies SVD to the original or sampled term-document matrix. Note that RP-eLSI and RP are different. RP directly applies random projection to the term-document matrix, whereas RP-eLSI only uses random projection as one substep of our eLSI algorithm to reduce the dimensionality of the centroid vectors.

4.3 Experimental Results

In this section, we evaluate eLSI and the above algorithms. Experiments are conducted with the TREC 7&8 corpus and queries.

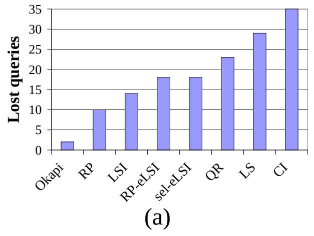
4.3.1 *Comparing Dimensionality Reduction Methods*

Figure 5 compares the dimensionality reduction methods. In this experiment, sel-eLSI and RP-eLSI use 2,000 centroids of 2,000 dimensions. We use semantic vectors generated by each method to retrieve 1,000 documents for a query and report the average number of retrieved relevant documents in Figure 5(a). The dimensionality of the semantic space varies from 20 to 300 (shown on the X axis), except for RP. RP's recall is among the worst when the dimensionality is low but we notice a big improvement when the dimensionality increases from 200 to 300. Out of curiosity, we further increase the dimensionality up to 900 and observe that RP keeps its momentum. This matches with the theory that RP performs well when the dimensionality of the reduced space is sufficient in capturing the real dimensionality of the data. When the dimensionality of the reduced space is insufficient, RP's recall degrades quickly because, unlike LSI, it does not try to capture the major structure of the data in low dimensions.

The recall of eLSI (both sel-eLSI and RP-eLSI) is very close to LSI, but we start to see a difference when the dimensionality is bigger than 140. eLSI uses document clustering plus term selection or RP to reduce the size of the input matrix for SVD. It retains the major structure of the data but does lose some fine details that are captured by the high-dimensional elements of the semantic vectors produced by LSI. pLSI only exploits information in low-dimensional subvectors (usually lower than 150 dimensions) to guide index placement and query routing (see Section 2.4). Therefore, when used with pLSI, the choice of LSI or eLSI makes no difference in retrieval quality. eLSI outperforms QR, LS, and CI. With a 140-dimensional space, eLSI retrieves 38%, 79%, and 102% more relevant documents than QR, LS, and CI, respectively. This is because eLSI uses more document clusters, which contain more information about the corpus, and SVD is superior than other methods in dimensionality reduction.

Figure 5(b) and (c) report the precision-recall for different methods in a 140-dimensional and 300-dimensional space, respectively. Since the performance of RP-eLSI and sel-eLSI are similar, we omit the results for sel-eLSI for clarity. To our surprise, although RP's recall is among the worst in Figure 5(a), its high-end precision is among the best in Figure 5(b) and (c). This indicates that, after random projection, vectors that are very close in the original high-dimensional space are still very close in the low-dimensional space. If two vectors have a medium or long distance in the original space, their distance in the low-dimensional space may be distorted, however. This is the reason why RP has a low recall. In the 140-dimensional space, eLSI's precision-recall is similar to that of LSI; in the 300-dimensional space, eLSI's performance is noticeably worse, which is consistent with the results in Figure 5(a).

Figure 6(a) reports the number of queries that find no relevant document in a 300-dimensional space. Consistent with the results



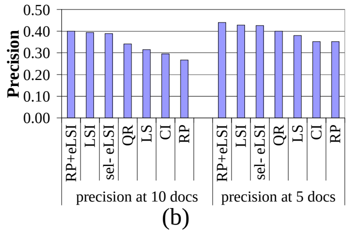


Figure 6: Comparison of dimensionality reduction methods. (a) Queries that find no relevant document. (b) High-end precision when combining with Okapi.

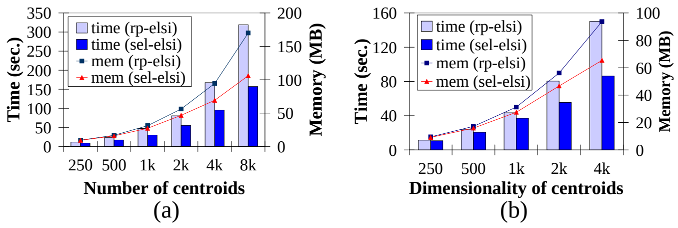


Figure 7: Efficiency of eLSI. Memory consumption and execution time of SVD when varying (a) the number of document centroids; and (b) the dimensionality of the reduced centroid matrix $\tilde{C}$.

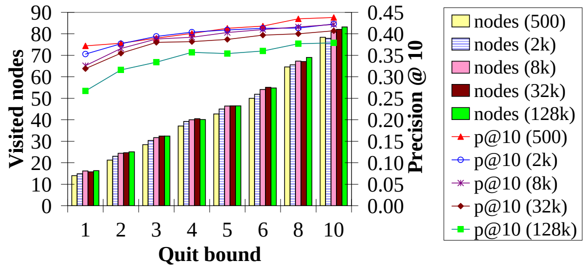


Figure 8: The number of visited nodes and precision at top 10 documents when varying the number of nodes in the system from 500 to 128,000 (in parentheses).

in Figure 5(b) and (c), RP performs best, which again indicates that RP is good at keeping distance between very close vectors.

Figure 6(b) reports the high-end precision when combining dimensionality reduction methods with Okapi. The configuration is similar to that for LSI+Okapi in Figure 3(c). We project documents into a 100-dimensional space and then partition the 100-dimensional space into four planes, each of which is of 25 dimensions. Each plane uses its subvectors to retrieve 1,000 documents for a query, resulting in a total of 4,000 returned documents. Finally, we use Okapi to rank these returned documents and report the precision at top 5 and 10 documents. When used with Okapi, eLSI achieves precisions almost identical to LSI and outperforms all other methods. RP's precision is the worst in this figure, which seems to contradict with previous results. However, since we use Okapi to rerank documents, what dimensionality reduction methods really contribute is their recall. RP's recall is the worst in spaces of very low dimensions (25 dimensions in this experiment). Therefore, the precision of RP with Okapi reranking is the worst.

In summary, when the goal is high recall or when the dimensionality reduction method is to be used with other ranking algorithms, we recommend eLSI. When the goal is high precision (without further ranking by other algorithms), we recommend RP.

4.3.2 *Scalability of eLSI*

The main computation in eLSI is document clustering and SVD. We consider clustering to be much more scalable than SVD. The time complexity of the clustering algorithm we use is $O(n \log(s))$, where n is the number of documents and s is the number of clusters. One can use more efficient algorithms based on data summarization or random sampling, or use algorithms that only scan the data set once. Clustering is inherently data parallel. Implemented properly, distributed clustering can achieve almost linear speedup. Interested readers may refer to [2] for all these details. A comparison of clustering algorithms is out of the scope of this paper. Our evaluations below will focus on the scalability of SVD.

Figure 7 reports the execution time and memory consumption of using SVD to project eLSI's reduced centroid matrix $\tilde{C}$ into a 150-dimensional space. For sel-eLSI, the dimensionality of $\tilde{C}$ is the number of terms kept in $\tilde{C}$ after term selection. For RP-eLSI, the dimensionality of $\tilde{C}$ is the reduced dimensionality after random projection. In Figure 7(a), we vary the number of document clusters while fixing the dimensionality of $\tilde{C}$ to 2,000. In Figure 7(b), we vary the dimensionality of $\tilde{C}$ while fixing the number of document clusters to 2,000. The cost of SVD scales reasonably well in both figures. sel-eLSI is more efficient than RP-eLSI because the matrix $\tilde{C}$ produced by sel-eLSI is more sparse than that produced by RP-eLSI. Recall that the cost of SVD is proportional to the number of nonzero elements in the input matrix to SVD. A cluster of similar documents tend to use a small vocabulary. Therefore the centroid vectors are sparse. When the TREC corpus is partitioned into 2,000 clusters, on average 98.8% of elements in a centroid are zero. After term selection, many elements of matrix $\tilde{C}$ are still zero. RP uses a random matrix for projection. After projection, the probability of having zero elements in matrix $\tilde{C}$ is very low. Since sel-eLSI is more efficient and the retrieval quality of RP-eLSI and sel-eLSI are similar, we opt for sel-eLSI.

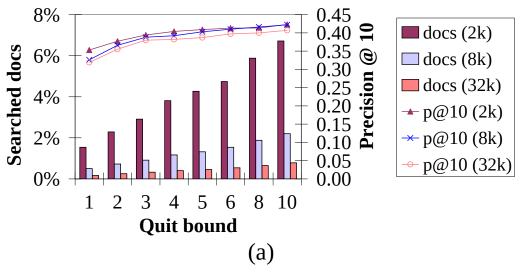
eLSI significantly reduces LSI's computation cost for SVD. SVD for sel-eLSI with a 2000-by-2000 centroid matrix $\tilde{C}$ takes 55 seconds and consumes 47MB memory, whereas running SVD over the sampled 15% TREC documents takes 57 minutes and consumes 1.7GB memory. The good retrieval quality of eLSI indicates that document clustering and term selection do keep important content of the term-document matrix. We believe that, when combined with an efficient clustering algorithm and a parallel implementation of SVD [18], eLSI can handle very large copora.

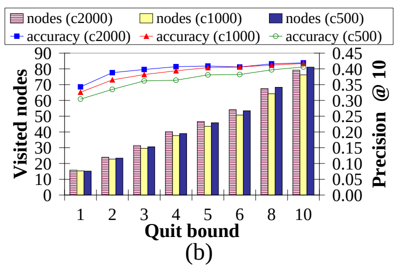
5. PERFORMANCE OF PSEARCH

We use the TREC 7&8 corpus and queries to evaluate the complete pSearch system that includes the enhancements for LSI we developed in Section 3 and 4. pSearch has some details not covered in this paper. Unless otherwise noted, we use the configuration in Table 1 of [32] for those omitted features. The semantic vectors are generated by "sel-eLSI" (below we will simply refer to it as "eLSI"). eLSI uses 2,000 document clusters and 2,000 selected terms. We use Okapi to guide content-directed search and to select documents on visited nodes.

Figure 8 shows the number of visited nodes and precision at top 10 documents when varying the number of nodes in the system from 500 to 128,000 (in parentheses). The X axis is the quit threshold T that controls the size of the search region (see Section 2.4). A search is terminated when no better document is found on the most recently searched T nodes. Okapi's precision at top 10 documents is 0.45. pSearch achieves a precision close to Okapi by visiting only a small number of nodes. This performance is scalable with respect to system size. For an 8,000-node system, pSearch achieves a precision of 0.4 by visiting 47 nodes. For a 32,000-node system, pSearch achieves a precision of 0.4 by visiting 67 nodes. It can achieve a higher precision by visiting more nodes. When searching a similar number of nodes, the precision decreases as the system size increases. Since the size of the corpus is constant, more nodes imply fewer indices on each node. Results not presented here show that search efficiency and retrieval quality improve as the number of indices stored on each node increases.

The experiment in Figure 9(a) is the same as that in Figure 8 but we report here the documents searched on those visited nodes as a percentage of the entire corpus. To some extent this reflects the balanced distribution of indices across nodes (see [32] for details on load balance). For clarity, we only vary node population from





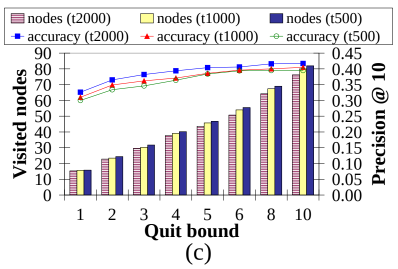


Figure 9: (a) Precision at top 10 documents and documents searched on the visited nodes as a percentage of the entire corpus, when varying the number of nodes in the system from 2,000 to 32,000 (in parentheses). eLSI's impact on pSearch's retrieval quality when (b) varying the number of document clusters; and (c) varying the number of selected terms.

2,000 to 32,000. For the 8,000-node system, pSearch searches only 1.3% of TREC to achieve a precision of 0.4 for top 10 documents.

Figure 9(b) and (c) evaluates the impact on retrieval quality when eLSI reduces the size of the input matrix for SVD through document clustering and term selection. This experiment uses a 10,000-node system. In Figure 9(b), we fix the number of selected terms to 2,000 while varying the number of document clusters from 500 to 2,000 (in parentheses). In Figure 9(c), we fix the number of document clusters to 1,000 while varying the number of selected terms from 500 to 2,000 (in parentheses). When reducing the size of the input for SVD dramatically, we only see a minor degradation in precision, indicating that eLSI is scalable. The performance gap between different input sizes diminishes as pSearch searches more nodes to achieve a higher precision.

Overall, pSearch is efficient and effective. It searches a small number of nodes to achieve a precision close to the state-of-the-art centralized baseline. pSearch is scalable with respect to system size. When the number of nodes increases exponentially, the number of visited nodes increases moderately and the precision degrades marginally. We expect it to be scalable with respect to corpus size as well, since eLSI can aggressively reduce the size of the input for SVD without seriously compromising the quality of semantic vectors it produces. The performance is expected to improve as the average number of indices stored on a node increases.

6. RELATED WORK

In pSearch, we adopted techniques from several fields to build an efficient P2P IR system. Due to space limitation, we will only introduce works that are most relevant. A survey of clustering algorithms can be found in [2]. Many methods have been proposed to reduce search space for multi-dimensional data [35]. Our use of a CAN to partition a Cartesian space is similar to Grid File [21]. To our knowledge, all existing multi-dimensional data access methods, including Grid File, are designed for centralized systems.

6.1 Distributed Information Retrieval

Centralized IR systems suffer from a single point of failure and performance bottleneck at the server. Flooding-based techniques such as Gnutella send a query to every node in the system, consuming huge amounts of system resources. To reduce this cost, heuristic-based approaches direct a search to only a subset of nodes. Rhea and Kubiatowicz [26] used attenuated Bloom filters to summarize content on neighbors of a node. A node only forwards a query to neighbors that are likely to contain relevant documents. GlOSS [13] uses a hierarchy of meta-databases to summarize content of other databases. During a search, the summary is referenced to choose databases that may contain most relevant documents.

Both [36] and [17] pointed out problems with conventional database selection algorithms when dealing with heterogeneous databases. They proposed to cluster database content into topics. Searches are conducted in topically organized databases. Database selection is guided by a centralized meta-database that maintains information about topics. SETS [1] follows a similar approach but floods the meta-database to all nodes. The number of topics in these systems are predetermined and they only route queries among a relatively small number of distributed sites. In contrast, pLSI uses semantic vectors to cluster documents in a large P2P overlay. It does not explicitly maintain the meta-database since this knowledge is embedded in the structure of the overlay network. The number of "topics" automatically adapts to current node population.

Some P2P search systems directly implement inverted files on top of a DHT [19]. Each node is responsible for the inverted lists for some terms. To answer multi-term queries, the inverted lists must be transmitted over the network such that an intersection to identify documents that contain multiple query terms can be performed. This communication cost grows with the corpus size. eSearch uses a novel hybrid global-local indexing structure to avoid this communication at the expense of moderate index replication [31].

6.2 Enhancements to LSI

Various fast dimensionality reduction methods have been proposed to approximate LSI. The three methods based on document clustering (CI, LS, and QR) [9, 15, 23] were originally proposed for document categorization. We found that, when used for document retrieval, their performance is not as good as our eLSI algorithm. Bingham and Mannila [6] reported reasonably good performance when using random projection (RP) to reduce the dimensionality of text data. They used a corpus that was more than 200 times smaller than the corpus we used. We found that when used alone to *aggressively* reduce dimensionality, RP's recall is the worst among eLSI, CI, LS, and QR. However, when used as one substep of eLSI, RP is effective when it reduces dimensionality not so aggressively. Papadimitriou et al. [22] used RP to reduce the dimensionality of *document* vectors, prior to applying SVD. Our RP-eLSI algorithm uses RP to reduce the dimensionality of *centroid* vectors, producing a much smaller input matrix for SVD. Moreover, they only presented experimental results on a set of artificially generated documents. Kolda and O'Leary [16] used semi-discrete matrix decomposition (SDD), instead of SVD, with LSI. Compared with SVD, SDD uses less memory but takes much more time. Potentially, one can use eLSI to reduce the size of the input for SDD.

Husbands et al. [14] found that normalizing semantic vectors for terms improves LSI's retrieval quality by emphasizing rare terms. We discovered that, for large corpora, it is important to normalize semantic vectors for both terms and documents. We observed performance discrepancies between a small, homogeneous corpus and a large, heterogeneous corpus and gave one explanation. That is, normalizations are beneficial when the dimensions of the semantic space are insufficient in capturing the fine structure of a corpus.

7. CONCLUSIONS

We have described several techniques to scale LSI to work in a large P2P system and quantified the efficiency and efficacy of pSearch by experimenting with the large TREC corpus. We made the following contributions in this paper.

- We proposed the eLSI algorithm to improve the efficiency of LSI. eLSI uses document clustering plus term selection or random projection to reduce the size of the input matrix for SVD, while retaining the matrix's major content. eLSI retains the retrieval quality of LSI but is several orders of magnitude more efficient. It outperforms four major fast dimensionality reduction methods in retrieval quality.
- Through extensive experimentation, we found that, when the dimensions of the semantic space are insufficient to capture the fine structure of the corpus, proper normalizations of semantic vectors for terms and documents improve recall by 76% compared with the standard LSI baseline.
- Without sufficient dimensions, LSI cannot accurately rank documents for large corpora. We found that LSI can still cluster documents properly because it captures the important structure of the corpus. Our LSI+Okapi algorithm combines the benefit of LSI and Okapi. It uses LSI to cluster documents in a P2P network to reduce the search space, but uses Okapi to guide the search process and document selection.

The combination of our optimizations makes pSearch both more efficient and more effective. For a 32,000-node system, pSearch's precision at 10 retrieved documents (for the TREC corpus) is 0.4 compared with Okapi's 0.45, when on average searching only 67 nodes for a query. Although pSearch's high-end precision approaches that of state-of-the-art centralized IR systems, its low-end precision is inferior. Our future work includes improving low-end precision, incorporating other IR techniques such as automatic query expansion and PageRank, and experimenting with a large corpus crawled from the Web.

8. REFERENCES

[1] M. Bawa, G. S. Manku, and P. Raghavan. SETS: Search Enhanced by Topic Segmentation. In *SIGIR'03*, 2003.

[2] P. Berkhin. Survey of clustering data mining techniques. Technical report, Accrue Software, San Jose, CA, 2002.

[3] M. Berry, Z. Drmac, and E. Jessup. Matrices, vector spaces, and information retrieval. *SIAM Review*, 41(2):335–362, 1999.

[4] M. W. Berry and M. Browne. *Understanding Search Engines: Mathematical Modeling and Text Retrieval (Software, Environments, Tools)*. Society for Industrial & Applied Mathematics, 1999.

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

[6] E. Bingham and H. Mannila. Random projection in dimensionality reduction: applications to image and text data. In *SIGKDD'01*, 2001.

[7] C. Buckley. Implementation of the SMART information retrieval system. Technical Report TR85-686, Department of Computer Science, Cornell University, Ithaca, NY 14853, May 1985. Source code available at ftp://ftp.cs.cornell.edu/pub/smart.

[8] 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.

[9] I. S. Dhillon and D. S. Modha. Concept decompositions for large sparse text data using clustering. *Machine Learning*, 42(1):143–175, 2001.

[10] S. Dumais. Using LSI for information filtering: TREC-3 experiments. In *Third Text REtrieval Conference (TREC-3)*, 1995.

[11] P. Frankl and H. Maehara. The johnson-lindenstrauss lemma and the sphericity of some graphs. *Journal of Combinatorial Theory Ser. B*, 44(3):355–362, 1988.

[12] G. Golub and C. V. Loan. *Matrix Computations*. The Jason Hopkins University Press, Baltimore, Maryland, second edition edition, 1989.

[13] L. Gravano, H. García-Molina, and A. Tomasic. GlOSS: text-source discovery over the Internet. *ACM Transactions on Database Systems*, 24(2), 1999.

[14] P. Husbands, H. Simon, and C. Ding. the use of singular value decomposition for text retrieval. In M. Berry, editor, *Proc. of SIAM Comp. Info. Retrieval Workshop*, October 2000.

[15] G. Karypis and E.-H. S. Han. Concept indexing: A fast dimensionality reduction algorithm with applications to document retrieval and categorization. In *CIKM'00*, 2000.

[16] T. G. Kolda and D. P. O'Leary. semidiscrete matrix decomposition for latent semantic indexing in information retrieval. *ACM Trans. Information Systems*, 16:322–346, 1998.

[17] L. S. Larkey, M. E. Connell, and J. P. Callan. Collection Selection and Results Merging with Topically Organized U.S. Patents and TREC Data. In *CIKM'00*, 2000.

[18] T. A. Letsche and M. W. Berry. Large-scale information retrieval with latent semantic indexing. *Information Sciences*, 100(1-4):105–137, 1997.

[19] J. Li, B. T. Loo, J. Hellerstein, F. Kaashoek, D. R. Karger, and R. Morris. On the Feasibility of Peer-to-Peer Web Indexing and Search. In *IPTPS'03*, February 2003.

[20] D. S. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu. Peer-to-peer computing. Technical Report HPL-2002-57, HP Lab, 2002.

[21] J. Nievergelt, H. Hinterberger, and K. C. Sevcik. The grid file: An adaptable, symmetric multikey file structure. *ACM Transactions on Database Systems*, 9(1):38–71, 1984.

[22] C. H. Papadimitriou, H. Tamaki, P. Raghavan, and S. Vempala. Latent Semantic Indexing: A Probabilistic Analysis. In *PODC'98*, 1998.

[23] H. Park, M. Jeon, and J. Rosen. Lower dimensional representation of text data based on centroids and least squares. *BIT*, 43(2):1–22, 2003.

[24] C. D. Prete, J. T. McArthur, R. L. Villars, I. L. Nathan Redmond, and D. Reinsel. Industry developments and models, Disruptive Innovation in Enterprise Computing: storage. *IDC*, February 2003.

[25] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-Addressable Network. In *SIGCOMM'01*, 2001.

[26] S. Rhea and J. Kubiatowicz. Probabilistic Location and Routing. In *INFOCOM'02*, 2002.

[27] S. E. Robertson, S. Walker, S. Jones, M. M. HancockBeaulieu, and M. Gatford. Okapi at TREC-3. In *TREC-3*, 1994.

[28] G. Salton, A. Wong, and C. Yang. A vector space model for information retrieval. *Journal for the American Society for Information Retrieval*, 18(11):613–620, 1975.

[29] A. Singhal, C. Buckley, and M. Mitra. Pivoted Document Length Normalization. In *SIGIR'96*, 1996.

[30] SVDPACK. http://www.netlib.org/svdpack.

[31] C. Tang and S. Dwarkadas. Peer-to-Peer Information Retrieval in Distributed Hashtable Systems. In *NSDI'04*, 2004.

[32] C. Tang, Z. Xu, and S. Dwarkadas. Peer-to-Peer Information Retrieval Using Self-Organizing Semantic Overlay Networks. In *SIGCOMM'03*, 2003.

[33] C. Tang, Z. Xu, and M. Mahalingam. pSearch: Information Retrieval in Structured Overlays. In *The First Workshop on Hot Topics in Networks (HotNets I)*, 2002. Older but partially expanded version available as technical report HPL-2002-198, "PeerSearch: Efficient Information Retrieval in Peer- to-Peer Networks".

[34] Text Retrieval Conference (TREC). http://trec.nist.gov.

[35] R. Weber, H.-J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *VLDB'98*, 1998.

[36] J. Xu and W. B. Croft. Cluster-Based Language Models for Distributed Retrieval. In *SIGIR'99*, 1999.