March 28, 1999
RT0300
Computer Science 31 pages

Research Report

Information Retrieval and Ranking on the Web: Benchmarking Studies I

Georges DUPRET and Mei KOBAYASHI

IBM Research, Tokyo Research Laboratory
IBM Japan, Ltd.
1623-14 Shimotsuruma, Yamato
Kanagawa 242-8502, Japan

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

Information Retrieval and Ranking on the Web: Benchmarking Studies I

Georges E. Dupret, gedupret@hotmail.com
Institute of Policy and Planning Sciences, University of Tsukuba
1-1-1 Tennoudai, Tsukuba-shi, Ibaraki 305 Japan
and
Mei Kobayashi, mei@trl.ibm.co.jp
IBM Research, Tokyo Research Laboratory, IBM Japan, Ltd.
1623-14 Shimotsuruma, Yamato-shi, Kanagawa-ken 242-8502 Japan

March 28, 1999

Abstract

The exponential growth of information available on the World Wide Web has been documented in numerous studies. The studies also indicate that Internet users are turning to search engines and search services in increasing numbers to find the information they are seeking, but they are not necessarily satisfied with their performance. Specific problems which have been cited in user surveys include the speed of transmission and retrieval of information and the format for presenting the results from searches. In this report, we describe some of the components of a new Web-based search and retrieval system prototype, which is part of a larger information outlining and visualization system for Web documents. Our system is based on a modified version of latent semantic indexing, the output from which is used to rank the relevance of Web pages for an input query. We report the speeds of computation of the singular values and ranking when Householder bidiagonalization and Givens rotations are used to compute the singular values of a matrix representation of document-query space. In particular, our discussions will emphasize mathematical algorithms, efficient dynamic memory allocation procedures, and relevance ranking; linguistic techniques will be discussed only as necessary for the sake of completeness.

keywords: information retrieval, Internet, latent semantic indexing, relevance ranking, search engine, World Wide Web, WWW, W3.

1 Introduction

The exponential growth of information available on the World Wide Web has been documented in numerous studies. Furthermore, these studies indicate that Internet users are turning to search engines and search services in increasing numbers to find the information they are seeking, but they are not necessarily satisfied with the performance of current search services. The speed of transmission and retrieval of information and the format for presenting results from searches are often cited as factors contributing to user dissatisfaction. In this paper, we describe some of the components of a new Web-based search and retrieval system prototype (outlined in Figure 1) which is part of a larger information outlining and visualization system for Web documents proposed in [Kobayashi et al. 1999]. Our system retrieves and ranks a user-specified number of relevant pages for an input query. We report the speed of computation associated with retrieval and ranking of some sample sets of Web pages for some test input queries.

This paper is organized as follows. In the remainder of this section, we present the motivation for and significance of our prototype system. In the second section, we review *latent semantic indexing* (LSI), an IR procedure which is based on a mathematical (matrix) model for document-query space [Deerwester et al. 1988], [Deerwester et al. 1990] before going on to describe our variation and extension of LSI. In the third section, we discuss dynamic data structures used in our large, sparse matrix and vector computations for IR. Our data structure is attractive for LSI for several reasons, such as its ability to accomodate different query input formats, ease in executing updates (i.e., additions and deletions of documents and query terms), and speeding-up of searching and ranking of relevant documents. Our programs include a small library of matrix and vector routines which exploit the structures to reduce computation and data access and retrieval times. In the fourth section, we discuss our implementation of a very simple version of LSI. The successful application of LSI to Web-based searches is contingent on the fast and accurate computation of the singular values of a matrix representation of document-query space. We studied the speed of computation and ranking when Householder bidiagonalization and Givens rotations are used to compute the singular value decomposition (i.e., the singular values and their associated singular vectors) of a matrix representation of document-query space. Results from our experiments are compared with those reported in [King, Kobayashi 1999], which use (1) inverse iteration and (2) the Lanczos method followed by Sturm sequencing. Our implementation of the singular value decomposition is much faster for small matrices, however, for larger matrices, the overhead associated with dynamic data structures is more than compensated for by the faster speed in computation. Our discussions emphasize efficient mathematical algorithms and dynamic data structures so that linguistic techniques are discussed only as necessary.

1.1 Motivation and Background

One of the keys to becoming the most popular and successful search engine is the development of new algorithms specifically designed for fast retrieval of valuable information on the Web. "*Speed*" (i.e., search engine search and retrieval time plus communication delays) has consis-

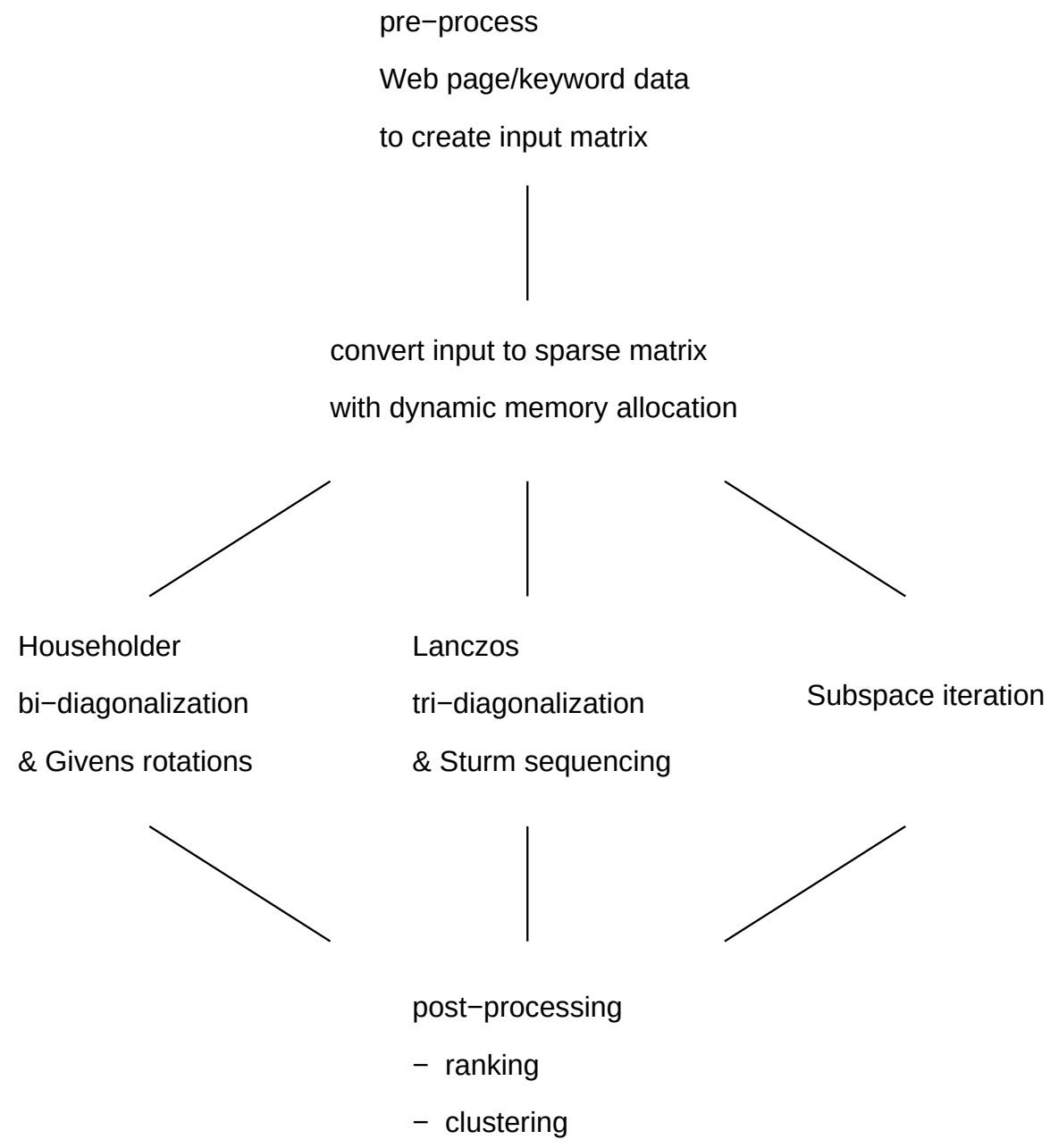


Figure 1: Overview of a Web document search and retrieval feature in an information outlining system

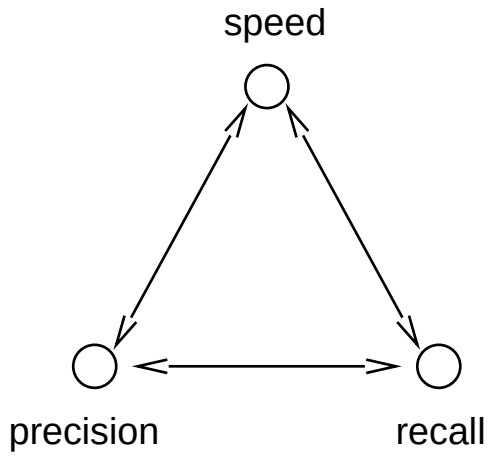


Figure 2: Three-way trade-off in search engine performance: (1) speed of retrieval, (2) precision, and (3) recall.

tently been cited as "*the most commonly experienced problem with the Web*" in the bi-annual WWW surveys conducted at the Graphics, Visualization, and Usability Center of Georgia Institute of Technology (GVU) [1] [GVU]. The tremedous growth in the number of Internet users and publicly accessible Web sites is making the problem with speed even more difficult to resolve. Although precise measurements of these numbers are difficult to make and verify, almost all studies report an exponential growth in both users and Web sites [Kobayashi, Takeda 1998].

Scientists are faced with a very delicate and difficult problem in developing Web search engines (illustrated in Figure 2): a three way trade-off between the speed of information retrieval, precision and recall becomes increasingly difficult to balance as the number of documents available through the Internet and number of users escalate. In the context of IR, *precision* is defined as the ratio of relevant documents to the number of retrieved documents, i.e.,

$$\text{precision} = \frac{\text{number of relevant documents}}{\text{number of retrieved documents}} ,$$

and *recall* is defined as the proportion of relevant documents that are retrieved, i.e.,

$$\text{recall} = \frac{\text{number of relevant, retrieved documents}}{\text{total number of relevant documents}} .$$

Speed, precision and recall are not the only factors contributing to user dissatisfaction with the Internet and Web. For example, Users have also cited dissatisfaction with input formats for queries, formats for presenting retrieved results, and the quality of retrieved information [Lawrence, Giles 1998]. A detailed discussion of ratings, with many references on search engines and their features is given in [Kobayashi, Takeda 1998]. Undoubtedly, the problems we have cited (and more !) are serving as an impetus for the tremendous amount of research on Web-based technologies.

[1] The GVU user survey appears to be one of the more reliable sources on user data. Its reports have been endorsed by the World Wide Web Consortium (W3C) and INRIA. It has recently come to be supported by a corporate council.

In this report, we describe some solutions to the problem associated with speed of retrieval (dynamic data structures for storing and accessing large, sparse matrices and vectors for computations and an implementation of LSI). The work described in this report and in [King, Kobayashi 1999] serves as the basis for a Web-based information retrieval component in a larger Web information outlining system. Information outlining systems retrieve, then format the retrieved results in a more attractive and meaningful manner, for example, as a graph, or a color-coded map [Morohashi et al. 1995], [Takeda, Nomiyama 1997]. The visual interface allows users to "*gain a better global understanding of the contents of digital libraries and navigate more easily through ... information*".

2 Latent Semantic Indexing

The study of IR from data bases has such a long history, particularly for text-based documents, that it is impossible to give a complete review. In this section, we give references to some works which are closely related to ours. Details of numerical algorithms and comparison of retrieval benchmarks (when appropriate and fair) will be given in Section 4. A broader review with selected references to many of the topics covered in this report is [Kobayashi, Takeda 1998].

Although *latent semantic indexing* (LSI) was originally developed as a document retrieval method for very large, isolated data bases, with modifications and enhancements, it shows promise for application to Web document databases. The original LSI algorithm automatically indexes and retrieves documents by mathematically modeling user queries and documents by vectors, and their associations by matrices [Deerwester et al. 1990][2]. The method has since been enhanced with the addition of differential term weighting and iterative retrieval methods [Dumais 1991]. The inventors of LSI claim that their algorithm overcomes deficiencies of earlier retrieval algorithms, such as hierarchical classification analysis (for term and document clustering) and factor analysis:

> "*hierarchical clusterings are far too limited to capture the rich semantics of most document sets. (They) permit no cross classifications, and in general have very few free parameters. ... Empirically, clustering improves the computational efficiency of search; whether or not it improves retrieval success is unclear ...*"
>
> – [Deerwester et al. 1990].

The view is shared by other experts, e.g., [Jardin, van Rijsbergen 1971], [Salton, McGill 1983], [Voorhees 1995]. Three of the more serious problems associated with factor analytic approaches are:

- high computational expense,

[2]The inventors have been issued a patent for the basic algorithm, however Bellcore (their employer at the time of filing) is the owner [Deerwester et al. 1988].

- difficulty/impossibility of implementation of all but low-dimensional representations, and
- "*the need for tedious data gathering techniques, requiring collection of thousands of similarity judgements from humans*" [Borko, Bernick 1963], [Ossorio 1966].

According to the inventors of LSI, the three strengths of the algorithm are:

- it involves a high-dimensional representation, which allows one to better represent a wide range of semantic relations;
- both terms and text objects are explicitly represented in the same space; and
- objects can be retrieved directly from query terms

[Deerwester et al. 1990]. In the remainder of this section, we present the LSI algorithm and discuss some mathematical techniques, methods to improve computational efficiency, and memory requirement issues which arise during its implementation.

2.1 The LSI Algorithm

In LSI, the relationship between query terms and documents in a data base are represented by an m-by-n matrix A, with ij-th entry a_{ij}, i.e.,

$$A = [a_{ij}].$$

The entries a_{ij} consist of information on whether term i occurs in document j and may also include weighting information to take into account factors, such as:

- the length of the document;
- the importance (or relevance) of the query term in the document; and
- the frequency of appearance of the query term in the document.

$A = [a_{ij}]$ is usually a very large, sparse matrix, because the number of (keyword) terms in any single document is usually a very small fraction of union of the (keyword) terms in all of the documents.

2.1.1 Weighting the Query-Document Matrix

In most implementations of LSI, both a local weighting term L_{ij} and global weighting term $G(i)$ are applied to a_{ij} [Letsche, Berry 1997]. Examples of local weighting functions which are simple and inexpensive to implement in an LSI scheme are: binary weighting, term-frequency weighting, and log-entropy weighting. Examples of global weighting functions which are simple and inexpensive to implement in an LSI scheme are: normal, GfIdf, Idf, and entropy weighting [Dumais 1991]. Also reported in [Dumais 1991] are observations that log-entropy weighting schemes appear to lead to particularly good results in LSI experiments.

When weighting is used in LSI, the local and global weighting functions $L(i,j)$ and $G(i)$ are applied to each (non-zero) element a_{ij} of A to generate a new matrix

$$\tilde{A} = [\tilde{a}_{ij}], \tag{1}$$

with entries

$$\tilde{a}_{ij} = L(i,j) \cdot G(i) \cdot a_{ij} \ . \tag{2}$$

The value of $\tilde{a}_{ij}$ can be further adjusted to take into account the frequency of occurrence of a query term by using, for example, a small multiplicative term ϵ_1 ; $0 \leq \epsilon_1 \leq 1$ for a frequency of 1 to 3, a medium size multiplicative term ϵ_2 ; $0 \leq \epsilon_1 \leq \epsilon_2 \leq 1$ for a frequency of 4 to 6, and a mutliplicative factor of unity for a frequency of 7 or greater.

2.1.2 The Singular Value Decomposition

The next step of the LSI algorithm, after the creation and weighting of matrix A, is the computation of the singular value decomposition (SVD)

$$A = U \Sigma V^T \tag{3}$$

(illustrated in Figure 3). To simplify notation, hereafter, we will refer to the document-query matrix as A, whether or not a weighting procedure, such as the one described above in equations (1) – (2) has been employed. Proof of the existence of the SVD for any matrix A, is given in numerous texts, including [Golub, Van Loan 1996] (Chapter 2, section 2.5.2, page 70):

Theorem: (Singular Value Decomposition, existence of): *If A is a real, m-by-n matrix, then there exist orthogonal matrices*

$$U = [u_1, u_2, \ldots, u_m] \in \Re^{m \times m} \ ,$$

and

$$V = [v_1, v_2, \ldots, v_n] \in \Re^{n \times n} \ ,$$

such that

$$U^T A V = \mathrm{diag}(\sigma_1, \sigma_2, \ldots, \sigma_p) \in \Re^{m \times n} \ ,$$

where $p = \min\{m,n\}$, and $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_p \geq 0$.

The columns of U and V are called the *left* and *right singular vectors*, and Σ is a diagonal matrix with monotonically decreasing diagonal elements σ_i, which are known as the *singular*

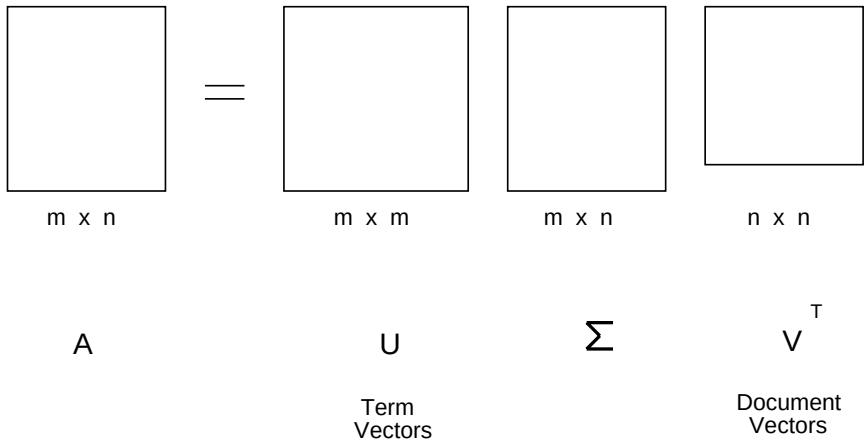


Figure 3: The singular value decomposition (SVD) of a matrix A.

values of the matrix A, i.e., Σ is of the form

$$\left[\begin{array}{ccccc|c} \sigma_1 & 0 & 0 & \cdots & 0 & \\ 0 & \sigma_2 & 0 & \cdots & \vdots & \\ & \ddots & \ddots & \ddots & & 0 \\ \vdots & & \ddots & \ddots & 0 & \\ 0 & \cdots & & 0 & \sigma_p & \\ \hline & & 0 & & & 0 \end{array}\right],$$

where $p \leq n$ and the matrix of zeroes on the top RHS is p-by-$(n-p)$, the matrix of zeroes on the bottom LHS is $(m-p)$-by-p, and the matrix of zeroes on the bottom RHS is $(m-p)$-by-$(n-p)$.

The classical paper on the SVD [Golub, Kahan 1969] is written as a mathematical research paper. Less formal, textbook type presentations for a more general audience are widely available in texts, including [Coleman, Van Loan 1988], [Golub, Van Loan 1996], [Jennings, McKeown 1979], [Press et al. 1982] and [Watkins 1991]. Studies of linear algebra techniques and their applications to IR include [Berry et al. 1995b], [Berry, Fierro 1996], and [Letsche, Berry 1997]. A comprehensive review and tutorial on using the SVD for IR is [Berry et al. 1995a], and an interesting review paper on the history of the SVD is [Stewart 1992].

The LSI algorithm reduces the noise in matrix A by contructing a modified matrix A_k, from the k largest singular values and their corresponding vectors, i.e.,

$$A_k = U_k \Sigma_k V_k^T \ ,$$

where Σ_k is a diagonal matrix with monotonically decreasing diagonal elements σ_i. The matrices U_k and V_k are the matrices whose columns are the left and right singular vectors of the k largest singular values of A (as shown in Figure 4). A_k is the closest rank-k approximation to A, in the least squares sense, as shown below in a theorem [Eckart, Young 1939].

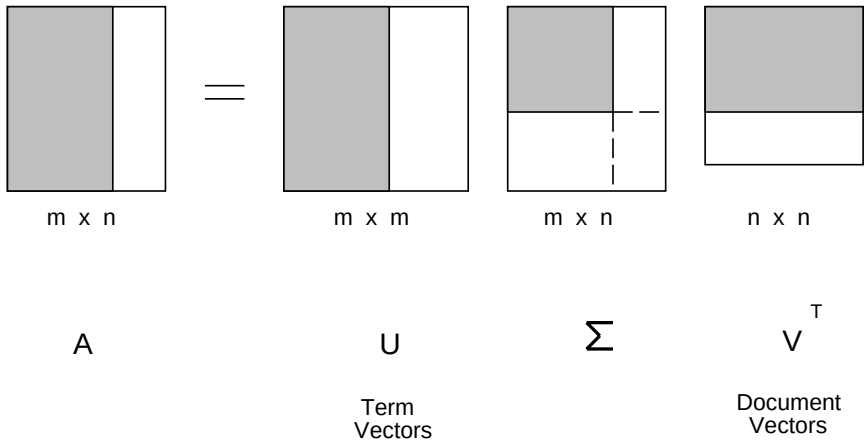


Figure 4: Construction of A_k, the closest rank-k approximation to A, through modification of the singular value decomposition of A.

Theorem (Eckhart and Young): *Let the singular value decomposition of A be given by equation (3) with $r = \mathrm{rank}(A) \leq p = \min(m,n)$, and define*

$$A_k = U_k \Sigma_k V_k^T \ ,$$

then

$$\min_{rank(B)=k} \|A - B\|_F^2 = \|A - A_k\|_F^2 = \sigma_{k+1}^2 + \cdots + \sigma_p^2 \ .$$

The proof is available in several texts, including [Golub, Van Loan 1996] and [Watkins 1991].

An important consideration in using the SVD is the sensitivity of singular values of a matrix A to small perturbation in each of the individual entries a_{ij}. Perturbation theory for the SVD is well documented, see, e.g., [Golub, Van Loan 1996] (Chapter 8, Section 8.6.1). We quote some the theorems from [Golub, Van Loan 1996] which are relevant for the SVD in LSI.

Theorem: *If $A \in \Re^{m \times n}$, then for $k = 1, 2, \ldots, \min\{m,n\}$,*

$$\begin{aligned} \sigma_k(A) &= \max_{\dim(S)=\dim(T)=k} \ \min_{x \in S, y \in T} \frac{y^T A x}{\|x\|_2 \|y\|_2} \\ &= \max_{\dim(S)=k} \ \min_{x \in S} \frac{\|Ax\|_2}{\|x\|_2} \ . \end{aligned}$$

Note that $S \subseteq \Re^n$ and $T \subseteq \Re^m$ are subspaces.

Theorem: *If A and $A + E$ are n-by-n symmetric matrices, then*

$$|\lambda_k(A + E) - \lambda_k(A)| \leq \|E\|_2 \ ,$$

for $k = 1, 2, 3, \ldots, n$.

Corollary: *If A and $A+E$ are in $\Re^{m\times n}$, with $m \geq n$, then*

$$|\sigma_k(A+E) - \sigma_k(A)| \leq \sigma_1(E) = \|E\|_2 \ ,$$

for $k = 1, 2, 3, \ldots, n$.

Proof: *Apply the theorem just above to*

$$\begin{bmatrix} 0 & A^T \\ A & 0 \end{bmatrix}$$

and

$$\begin{bmatrix} 0 & (A+E)^T \\ (A+E) & 0 \end{bmatrix}.$$

Although the perturbation theorems for the SVD give us upper bounds on the changes in the singular values, it is not known how the results will ultimately affect the quality of results retrieved from searches using LSI. The quality can only be determined through implementation studies.

2.1.3 Query Projection and Matching

Processing the query in LSI-based IR takes place in two steps: *query projection* followed by *matching* [Letsche, Berry 1997]. In the query projection step, input queries are mapped to *pseudo-documents* in the reduced query-document space by the matrix U_k, then weighted by the corresponding singular values σ_i from the reduced rank, singular matrix Σ_k. The process can be described mathematically as

$$q \ \longrightarrow \ \hat{q} = q^T U_k \Sigma_k^{-1} \ ,$$

where q represents the original query vector and $\hat{q}$ the pseudo-document. In the second step, similarities between the pseudo-document $\hat{q}$ and documents in the reduced term document space V_k^T are computed using any one of many similarity measures, such as those discussed in [Kobayashi, Takeda 1998].

2.2 Implementation of LSI

Successful implementation of LSI depends on the proper use of linear algebra algorithms and techniques, particularly in the implementation of the SVD. There are many different ways to compute the SVD of a matrix. Evaluation of an algorithm depends on the specifics of the intended application and the available computing environment. In addition to straightforward considerations, such as the availability of computing and memory resources, attention should be given to data structures since they can facilitate or encumber the updating of information in the document-query matrix. Updating the matrix involves additions and deletions of rows and columns as well as updating specific matrix entries.

In Web-based IR, the document-query matrix is often enormous and very sparse, so pre-processing of the matrix is necessary to speed up computations. Pre-processing may change the degree of its sparsity and the size of the matrix to be considered for the SVD. For example, if all of the entries of one column (or row) of the matrix are zero or very, very small, the column (or row) might be eliminated during pre-processing since the document (or query term) is likely to have very little correlation with query terms (or documents). Also, if all of the entries of one column (or row) of a matrix are unity or close to unity, then the query term is a word found in a typical *stoplist* or the document is a Web page which is used for keyword *spamming*.

Stoplists are lists of words which occur very frequently and which give little information, if any, about the contents of a document. For example, the two most frequently occurring words, *the* and *of*, comprise 10% of words in an average English document; the next four most common words, *and*, *to*, *a* and *in*, comprise another 10%; and eighteen words (including the six above) account for about 30% of the words in an average document. Astute use of stoplists can reduce the size of a document by 30% to 50%. More detailed discussion on stoplists and other pre-processing algorithms associated with signature files can be found in Chapter 5 in [Korfhage 1997], Chapter 2 of [van Rijsbergen 1979], and [Fox 1992].

Spamming is a new phenomenon which appeared with the introduction of search engines, automatic indexers, and filters on the Web [Flynn 1996], [Liberatore 1998]. Its primary intent is to outsmart automated software systems for a variety of purposes. Spamming has been used as an advertising tool by entrepreneurs, cult recruiters, Web page authors seeking fame, and technically well-versed, but off-balanced individuals with a warped mentality similar to those of computer virus creators. A famous example of hidden text spamming was the embedding of words in a black background by the Heaven's Gate Cult, a technique which has come to be known as *font color spamming* [Liberatore 1998]. (Although the cult no longer exists, the Heaven's Gate home page is archived at the sunspot.net site: http://www.sunspot.net/news/special/heavensgatesite) We note that the term *spamming* has a broader meaning, related to the receiving of excessive amount of email or information. An excellent, broad overview of the subject is given in [Cranor, LaMacchia 1998]. In the context we are considering, the more specialized terms *spam-indexing*, *spam-dexing*, or *keyword spamming* are more precise.

The degree of sparsity and size of the post-processed (or post-filtered) matrix may affect the choice of the SVD algorithm. Another consideration in selecting the computational algorithm are output requirements, such as the number of singular values and singular vectors which must be computed and their degree of accuracy. Some algorithms require that all of the singular values be computed. For other algorithms, the computation amounts to constructing (either implicitly or explicitly) the symmetric, positive definite matrix B, where

$$B \;=\; A^T \cdot A \qquad \text{or} \qquad B \;=\; A \cdot A^T \;.$$

Since rank $(A^T \cdot A)$ $=$ rank $(A \cdot A^T)$, B is set to be the matrix with fewer rows and columns. The eigenvalues λ_i of B are the square of the singular values σ_i of A, that is,

$$\lambda_i = \sigma_i^2 \;.$$

If only the largest few or only the smallest few eigenvalues (and possibly the corresponding singular vectors) need to be determined, then it may be more efficient to use this second type of algorithm. The recently revised classic text on the symmetric eigenvalue problem [Parlett 1998] is recommended for readers interested in details of this approach.

In LSI, the number k of singular values and their corresponding singular vectors which need to be explicitly computed is usually quite small, i.e., at most a few hundred. Optimizing the choice of k through experiments and analysis of the data, can significantly reduce overall computational costs. The trade-off between computational expense, volume of retrieved information, and noise can be summarized as follows. When A_k has a large rank (corresponding to a large k, with significantly high associated computational costs), it increases the amount of information used during retrieval, however, if the dimension of A_k is equal to or almost as large as that of A, then most of the noise in A will be passed on to A_k, and the noise will contribute to poor results in retrieval.

We discuss the nuts and bolts of computing the SVD using several different algorithms in section 4. In particular, we discuss: Householder reflections followed by Givens rotations, the power method, the power method with Aitken's acceleration, subspace iteration (also called simultaneous iteration and orthogonal iteration), the basic Lanczos algorithm, Lanczos with full reorthogonalization, Lanczos with selective orthogonalization with and without modifications.

3 Dynamic Data Structures

Dynamic data structures are useful for reducing memory requirements for storage of and computations involving large, sparse matrices and vectors. Some computational and bookkeeping (i.e. memory) overhead is associated with these methods, so these methods only offer significant advantages when very large, sparse matrices are under consideration. That is, these methods will cost more in memory and computation time than straightforward methods when a matrix is small or dense. Since document-keyword matrices in IR are usually huge and very sparse, dynamic data structures can yield significant savings in memory and computational resources. However, details in implementing these structures must be carefully tuned to optimize results. In this section, we describe some dynamic data structures, but first, we review the most straightforward method for allocating memory for matrix operations which can be used for any matrix, including those which are dense.

3.1 Standard Memory Allocation

When a (possibly dense) matrix is stored in the most straightforward format found in textbooks, the time required to search and access its entries is of order one, but the memory requirement is relatively large, i.e., $M \times N$, where M is the number of rows in the matrix, and N is the number of columns. This mode of storage is not recommended for very large, sparse matrices, because the memory requirement may be prohibitive, and for most computational algorithms,

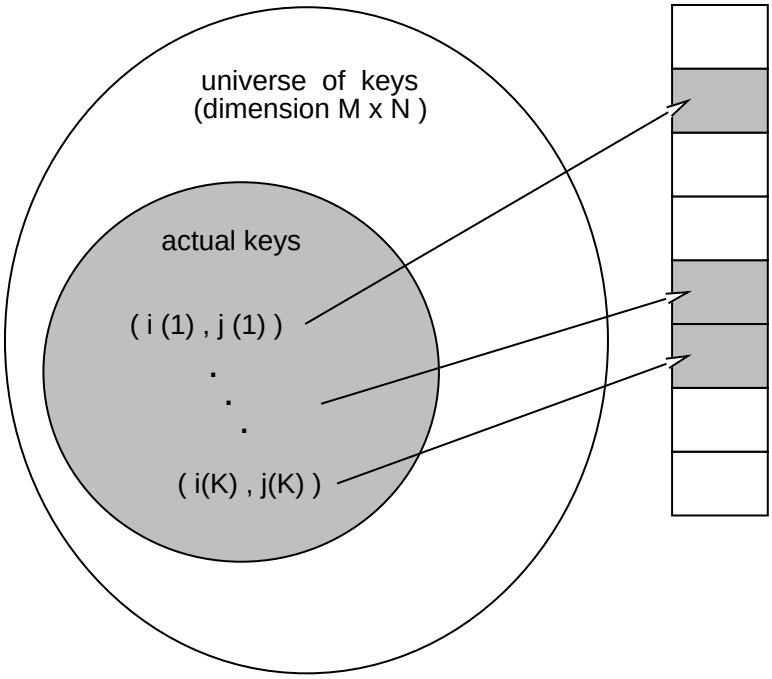


Figure 5: Hash tables

the associated speed is likely to be extremely slow, because of many unnecessary operations (e.g., addition, multiplication) with matrix entries which are zero.

3.2 Hashing

Hashing can significantly reduce the memory overhead associated with storage of and operations with large, sparse matrices. Hashing works as follows: Given an $M \times N$ matrix with s non-zero entries, where $s \ll MN$, determine a function, commonly known as a *hash function* h, which maps the universe of keys (of dimension MN) to a smaller space of dimension K (see Figure 5). To retrieve the value and indices of a non-zero entry of a matrix, compute its hash index using the hash function and retrieve the entry from the smaller space K. A very simple and commonly used hash function for an $M \times N$ matrix operates as follows: Given a non-zero entry in the (i, j) position, compute $[\ (i \cdot N) + j\ ](\bmod q\ )$, where q is an integer greater than 1. Unless q is extremely large, usually a problem known as collision arises (even when a matrix is extremely sparse). *Collision* is said to occur when two or more keys in the original universe of keys are mapped to the same key in the subspace of dimension K.

A variety of approaches can be used to resolve the collision problem. One well-known method, known as *collision resolution by chaining*, uses linked lists. A linked list consists of a sequence of blocks, where each block stores the value of a unique, non-zero entry of the matrix along with its indices and a pointer which gives the location of the next block of the list (see Figure 6):

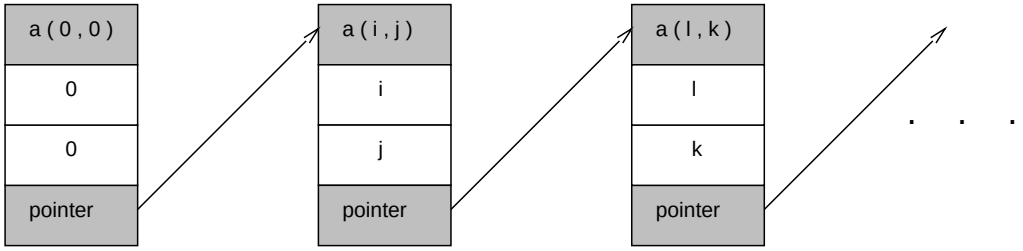


Figure 6: Collision resolution by chaining

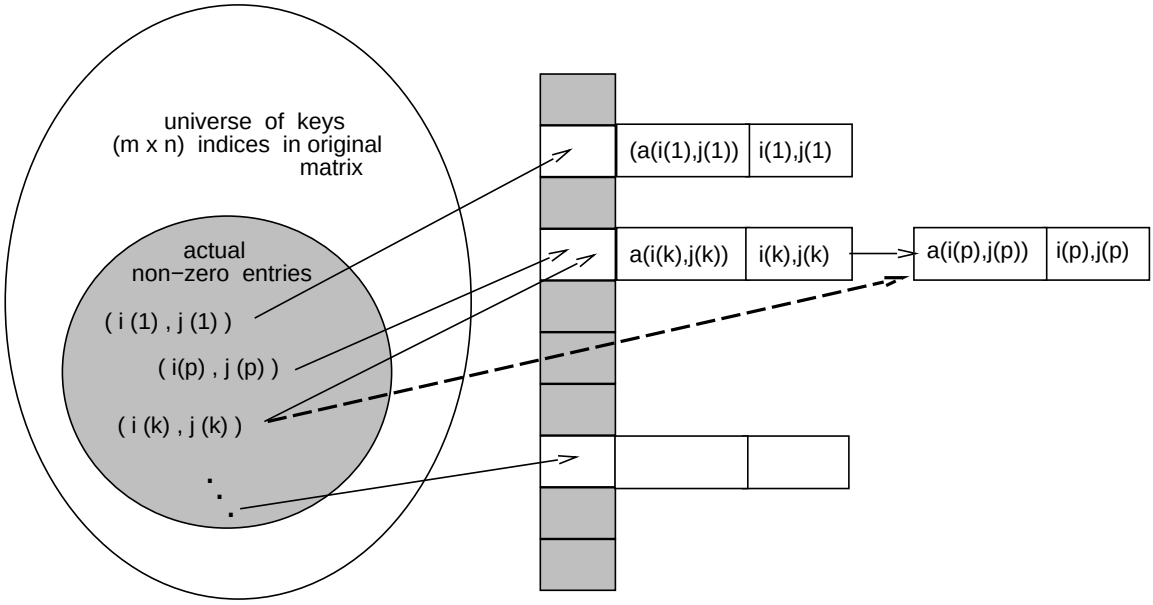


Figure 7: Storing Non-zero Elements

- **storage:** Each hash index is the origin of a linked list. In the absence of collision, each linked list has one or zero entries. But if collision occurs (i.e., information about a non-zero matrix entry is sent to a key already containing some entry) the new entry is added as an entry to the end of the appropriate linked list (see Figure 7).
- **retrieval:** To retrieve information about a matrix entry, its hash index is computed, and all of the entries in the corresponding linked list are examined to determine whether the entry is present. If the entry does not appear in the list, it is assigned the default value zero.
- **efficiency:** When the dimension of hash indices is K, and the mean number of entries per linked list is t, the average search time for an entry is of order $(1 + K)/t$.

A good hash function should distribute the information about the non-zero entries of a matrix as uniformly as possible in the smaller space of dimension K to minimize the average search time. The simple example given above, which uses a modulus function, is easy to implement and leads to fairly good results, in most cases.

3.3 Hashing: Our Implementations

In our data structures, the column index serves as a hash index. We create a vector of length N (where N is the number of columns), which consists of N pointers to N vectors. The n-th vector ($n = 1, 2, \ldots, N$) consists of information on the non-zero entries of the n-th column of the matrix. Information on non-zero entries in a particular column are stored in consecutive memory locations and information on the columns are, in turn, stored consecutively with respect to their index order (see Figure 8).

In LSI, each column corresponds to a document, and each row index to the index of a keyword so that efficient storage of the matrix is simple, because we can input the columns one-by-one with pre-sorted rows indices. In most general applications of hashing, entries are input one at a time, and linked lists do not necessarily preserve any kind of ordering of the matrix indices. In our LSI application, search times are expected to be shorter because the lists are ordered and matrix entries are placed in consecutive memory locations whenever possible. During the execution of our program, if a function call changes the memory requirements of a matrix (e.g., for copying operations, multiplication or entry search), then additional memory is allocated automatically or dynamically to meet the requirements.

More specifically, in our implementations, pointers to the addresses of columns are stored in consecutive blocks of memory. The address of each column points to a so-called *column object* composed of: (1) a consecutive block of memory with indices corresponding to the nonzero entries in the same column of the (very large and sparse LSI) matrix, and (2) a consecutive memory block with pairs of integers, i.e., the value of the non-zero entries in the column and the corresponding row index. Columns can be erased easily, added or swapped, because these operations involve changes to the pointers; the matrix entries, themselves, are not physically moved within the memory space.

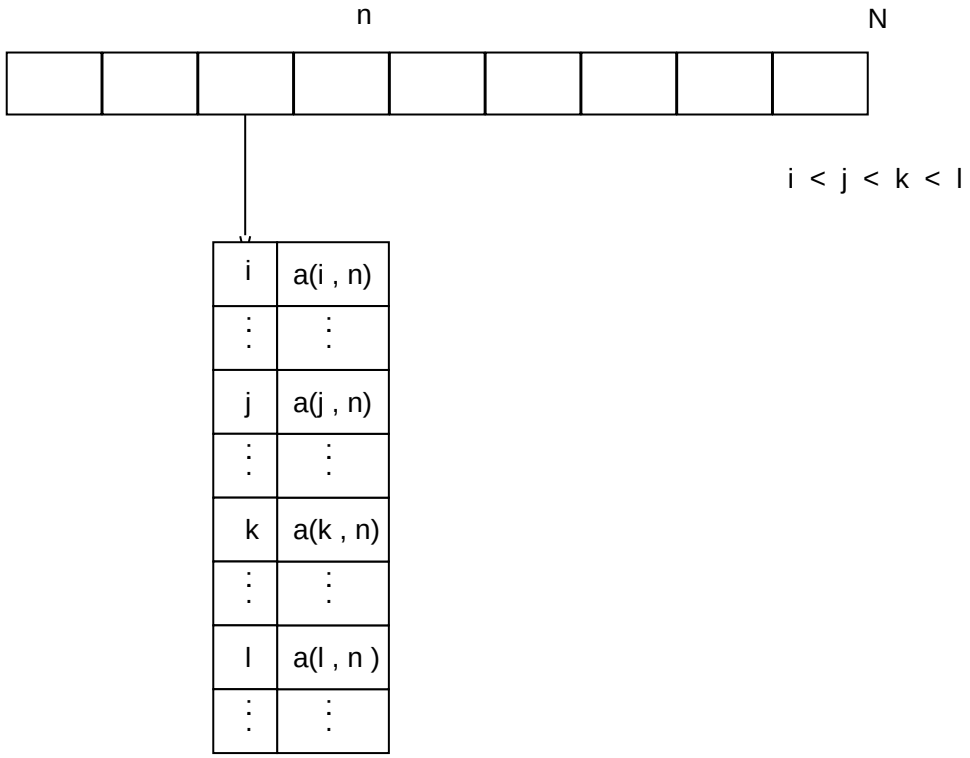


Figure 8: Storing Non-zero Elements

Pointers to addresses of rows are also stored in consecutive memory blocks. Each of these pointers addresses a *module*. A module contains: (1) an index corresponding to nonzero elements of the row, and (2) a pointer to the value of the nonzero element. Note that values of the nonzero elements have already been stored when our column access scheme (described above) was set up. Matrix entries can be easily accessed via rows, however, matrix updating via rows is more difficult than via columns. Addition of new columns is a simple task, however, deletion or swapping of columns requires substantial work, i.e., identifying the affected rows then updating the values. In fact, the cost of updating may well exceed any advantage(s) of having fast row access to matrix entries. Whether row accessing should be used depends on the intended application since it will determine which operations will be performed using the matrices.

We implemented our program so that we can access nonzero matrix entries by column or by row. Suppose we search for an entry by column: First, columns are accessed through pointers to their addresses. Next, the indices of nonzero entries in the column are scanned. And, finally, the value of the (nonzero) entry is returned. If the index of the desired element does not appear in the list of nonzero entries, then the value of the entry is zero. Searching by row is carried out in an analogous manner, but with one addtional step since modules have pointers to entries rather than the entries themselves.

In addition to reducing memory requirements, sparse matrix representation can facilitate fast computations with vectors and other matrices (both dense and sparse). Entries which

are non-zero are identified and arithmetic operations are carried out only when nonzero values coincide, i.e., unnecessary additions and multiplications involving zeros are not performed. The computational savings are slightly offset by the overhead of checking the indices of nonzero entries. As the sparsity of the matrix increases, the savings (in both computations and index checking) increase.

When our dynamic data structures are used to store matrices, vector-matrix multiplication can be carried out efficiently in two steps:

(1) determining the locations of non-zero entries a_{ij} of column j in the matrix (since the non-zero matrix entries of each column are stored consecutively in the memory along with their corresponding indices); and

(2) carrying out the multiplication of v_i with each non-zero a_{ij}; $j = 1, 2, \ldots, N$.

This two-step process minimizes the amount of time spent searching for and accessing the appropriate non-zero entries in the vector and matrix. Unfortunately, accessing non-zero entries by rows can be inefficient. One simple way to overcome this problem is to construct a vector row_i for each row, which contains information (i.e., pointers) to entries which are non-zero in that particular row (see Figure 9). The memory overhead associated with the addition of this feature is:

(1) a vector of length equivalent to the number of rows in the original matrix (each entry of the vector has a pointer to a new vector) plus

(2) the new vectors with indices of each of the non-zero elements in the row and pointers to the corresponding entry.

Our dynamic data structure is well-suited for Web-based LSI computations, because it is easy to add a new column to the matrix. Addition of a column to the document-query matrix in LSI corresponds to updating the matrix to introduce a new document.

4 Implementation Studies

In this section, we briefly review three methods for computing the SVD of a matrix:

- Householder reflections followed by Givens rotations,
- Power method/Inverse iteration, and
- Lanczos bidiagonalization followed by Sturm sequencing.

Scientitsts use these methods in many different applications to determine the top few hundred singular values and their associated singular vectors. Then, we present results from implementations studies in which the data are matrix representations of document-query space. For our implementations ing or small to medium-size matrices, we used Householder reflections followed by Givens rotations. Results from experiments using some large, sparse matrices and Lanczos methods are also given.

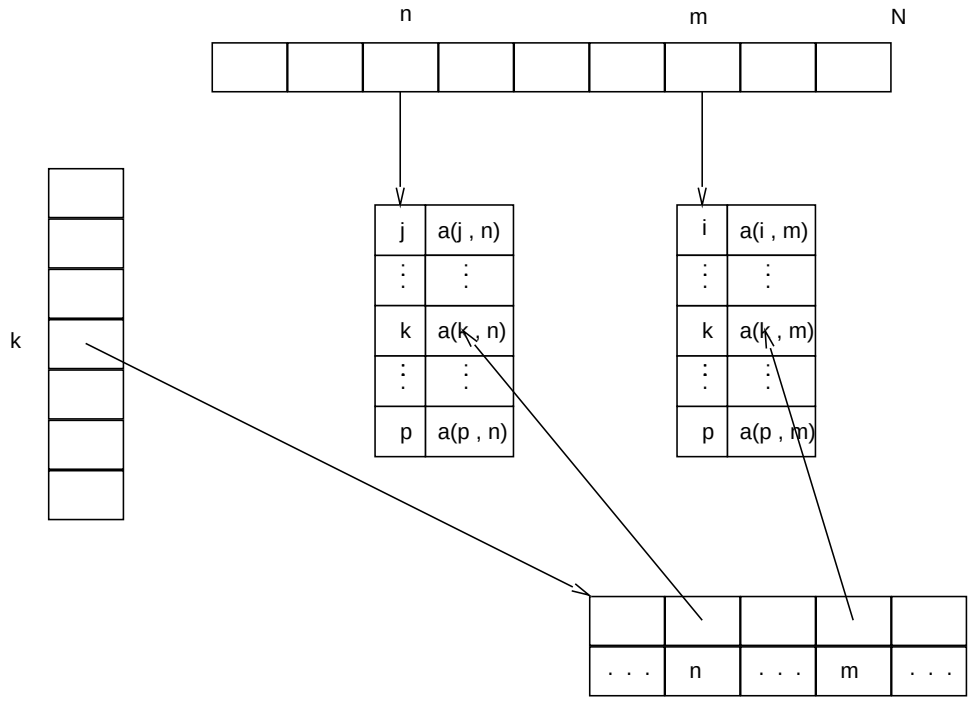


Figure 9: Storing Non-zero Elements

4.1 Householder Reflections and Givens Rotations

If A is quite small and no longer very sparse after post-processing, a viable approach is to use *Householder reflections* to bidiagonalize A, i.e., transform A to the form

$$\begin{bmatrix} \alpha_1 & \beta_1 & 0 & \cdots & 0 \\ 0 & \alpha_2 & \beta_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & \ddots & \beta_{n-1} \\ 0 & \cdots & \cdots & 0 & \alpha_n \\ * & \cdots & & \cdots & * \\ \vdots & & \ddots & & \vdots \\ * & \cdots & & \cdots & * \\ & & & & \end{bmatrix},$$

where $*$ denotes an entry, which may zero or nonzero. Next, apply sequences of plane rotators to zero the superdiagonal elements β_i. *Plane rotators* (also called *Givens rotators* and *Givens transformations*) are matrices which all non-diagonal entries are zero and diagonal entries are one. Exceptions occur on the i^{th} and j^{th} rows and i^{th} and j^{th} columns, in which

$$\begin{aligned} (i,i) &= (j,j) = \cos\theta \ , \\ (i,j) &= -(j,i) = -\sin\theta \ , \end{aligned}$$

where θ denotes the angle of rotation. Written out explicitly, *Givens rotators* are matrices which have the form

$$\begin{bmatrix}
1 & & & & & & & & & & \\
& 1 & & & & & & & & & \\
& & \ddots & & & & & & & & \\
& & & 1 & & & & & & & \\
& & & & c & \cdots & -s & & & & \\
& & & & & 1 & & & & & \\
& & & & \vdots & \ddots & \vdots & & & & \\
& & & & & & & 1 & & & \\
& & & & s & \cdots & c & & & & \\
& & & & & & & & 1 & & \\
& & & & & & & & & \ddots & \\
& & & & & & & & & & 1
\end{bmatrix},$$

where c denotes $\cos\theta$ and s denotes $\sin\theta$. Note that when the rotator is a 2-by-2 matrix, it reduces to the standard rotation matrix in a 2-dimensional plane. Details on Givens rotations can be found in standard texts, such as p. 142 in [Watkins 1991] and Chapter 5 of [Golub, Van Loan 1996].

When Givens rotations are used to zero an entry on the superdiagonal, it usually creates a new non-zero entry on the subdiagonal. For instance, zeroing the (1,2) entry causes re-computation of the (2,1) entry. When another Givens rotation is used to zero the new subdiagonal (2,1) element, a new nonzero entry is (usually) created in the super-super diagonal (1,3) entry. This process of using a sequence of Givens rotations to eventually remove each superdiagonal entry of a bi-diagonal matrix is called "*chasing*" or "*zero chasing*" (see Figure 10) [Lawson, Hanson, 1974]. A sequence of Givens rotations or zero chasing must be performed to zero each $(i, i+1)^{th}$ element of the bidiagonal matrix.

Using Householder transformations followed by Givens for computing the SVD will normally destroy much of the sparsity in the matrix A, require significantly more memory space and be computationally slower. Furthermore, this approach is far too slow for Web-based IR. We note that most of the older, over-the-counter software packages for computing the SVD were designed for the purpose of solving least squares problems, so they use Householder plus Givens approach, e.g., the book [Press et al. 1982] (software available on a diskette).

4.2 Power Method and Subspace Iteration

If A is no longer extremely large after pre-processing and is very sparse, a reasonable approach for computing the SVD of A for LSI may be subspace iteration followed by modified Gram-Schmidt. *Subspace iteration* is based on the *power method* – an even simpler algorithm, which

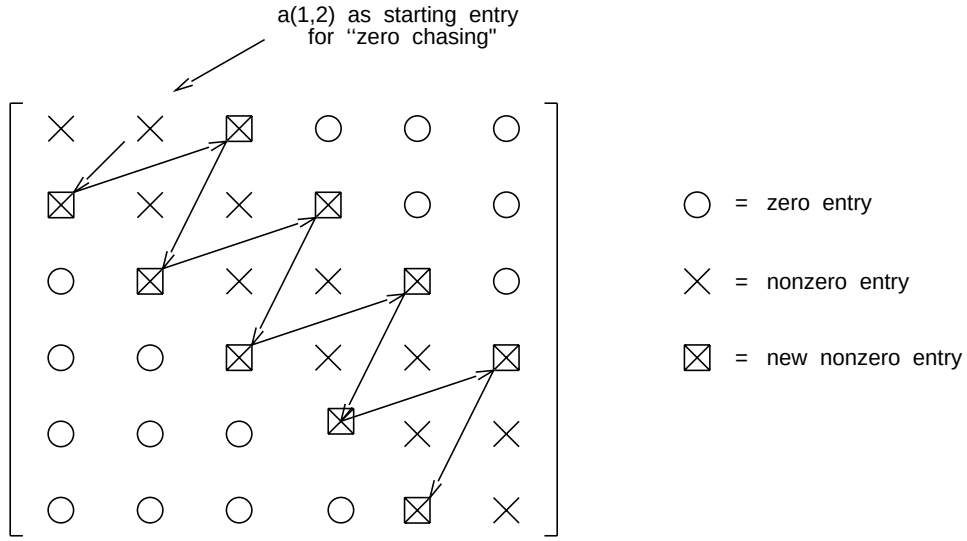


Figure 10: Example of "*zero chasing*" using Givens rotations on a 6-by-6 bidiagonal matrix.

is used in many scientific applications to determine the largest eigenvalue and the associated eigenvector of a matrix A. In both the power and subspace iteration methods, we consider the matrix products

$$B \;=\; A^T \cdot A \qquad \text{and} \qquad B \;=\; A \cdot A^T \;,$$

then set the smaller of the matrices to be B. The eigenvalues λ_i of B are the square of the singular values σ_i of A, i.e., $\lambda_i = \sigma_i^2$. Eigenvalue determination for our problem is not as difficult as for general matrices. Since B is symmetric, positive semidefinite, its eigenvalues are real, and all of its Jordan boxes are 1-by-1. In general eigenvalue finding programs, a substantial portion of extra code is devoted to tests for determining the (possible) existence of multiple roots and the size of associated Jordan boxes. And it is very difficult to write fail-safe, fast code which proceeses multiple and very very close roots.

In the *power method*, we begin with an arbitrary vector v of unit length and hope that the vector has a non-trivial component in the direction of the eigenvector associated with the largest eigenvalue. Then we compute the limit of the *Rayleigh quotient* of the matrix B, defined as

$$\lambda_1 = \lim_{m \to \infty} \frac{v^T B^{m+1} v}{v^T B^m v}$$

(details can be found in elementary numerical analysis texts, such as [Conte, de Boor 1980]). This computation can be reduced to matrix-vector and vector-vector multiplications to avoid explicit matrix-matrix multiplications. More specifically, if $B = A^T \cdot A$, the $v^T B v$ is computed as follows:

$$v^T B v \;=\; v^T (A^T (A v)) \;.$$

Similarly, multiplication begins from the rightmost vector and matrix when $B = A \cdot A^T$ [3]. One way to determine the second largest eigenvalue is to select a starting vector of unit length with no component in the direction of the eigenvector v_1, corresponding to the largest eigenvalue λ_1. Subsequent eigenvalues λ_n, can be determined by using a starting vector with no component in the directions of the $(n-1)$ largest eigenvectors $v_1, v_2 \ldots, v_{n-1}$, corresponding to the $(n-1)$ largest eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_{n-1}$. As many eigenvalues as desired can be computed this way, in theory, however, this "sequential" approach is not used in practice for determining multiple eigenvalues. The standard practice is to compute the desired number of eigenvalues simultaneously, using subspace or simultaneous iteration followed by modified Gram-Schmidt to ensure orthogonality of the recovered eigenvectors (details can be found in Section 5.2 in [Watkins 1991], Chapter 9 in [Jennings, McKeown 1979] or [Golub, Van Loan 1996]). Use of the modified, rather than clasical, Gram-Schmidt is recommended since numerical roundoff often leads to poor results when the classical method is used.

4.3 The Lanczos Algorithm

A good algorithm for computing some (but not all) of the singular values and the associated singular vectors of a large, sparse matrix A is to apply Lanczos tridiagonalization to the square matrix $B = A^T A$. Note that B should be computed implicitly to minimize the use of memory. Since B is symmetric, positive definite, Lanczos tridiagonalization will convert it to the form

$$\begin{bmatrix} \alpha_1 & \beta_1 & 0 & \cdots & 0 \\ \beta_1 & \alpha_2 & \beta_2 & \ddots & \vdots \\ 0 & \beta_2 & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \beta_{n-1} \\ 0 & \cdots & 0 & \beta_{n-1} & \alpha_n \end{bmatrix}$$

without many of the difficulties associated with the Lanczos method for more general matrices. A fast, reliable and parallelizable, eigenvalue routine, such as the Sturm sequence method can be used to compute the eigenvalues of B. Unfortunately, the associated eigenvectors must be computed sparately. Concise references to the algorithms are given in [Golub, Van Loan 1996], [Jennings, McKeown 1979], [Parlett 1998], and [Watkins 1991]. Extensive, detailed coverage of many variations of the Lanczos algorithm, depending on the properties of the input matrix is a two volume set [Cullum, Willoughby 1985]. The theory is given in the first volume and programming code in an outdated version of FORTRAN in the second volume.

Specialized software packages designed for computing the SVD of very large matrices using the Lanczos algorithm are: the subroutine SSVDC in LINPACK (further information is given in the users' guide [Dongarra et al. 1979], and a handbook [Coleman, Van Loan 1988]), LANSO [LANSO], and LAPACK and ScaLAPACK [Anderson et al. 1995], [Blackford et al. 1997],

[3] If we just want $v^T Bv$, we would be better off computing the value by taking the dot product $Av \cdot Av$ (which would reduce the work by one matrix-vector multiplication), however, we would like to know the value of Bv.

[Demmel 1997]. SVDPACK and SVDPACKC are two Lanczos software packages which have been used extensively for IR [Anderson et al. 1995], [Berry et al. 1993]. These packages are specifically designed to minimize numerical operations and use of memory, however, installing the package and understanding the user input and output interfaces may be difficult for inexperienced users. Details of our implementations of some variations of the Lanczos algorithm are given in [King, Kobayashi 1999].

4.4 Experimental Results

In our studies, we used matrices constructed from information in the Japanese newspaper *Nikkei* (1994) and ran our tests on an IBM RS/6000 machine. Results from our numerical experiments using Householder bidiagonalization and Givens rotations algorithms (described in Section 2) and dynamic data structures (described in Section 3) are given in Table 1 and Figure 11.

Table 1: Householder Reflections and Givens Rotations

matrix size	computation time
350	300
3600	41900
4700	59900
5100	66900
6500	99700
23900	345100
32250	1064900
48600	2236700
84000	2399700
126750	4848500

We subsequently ran experiments using the same data, but with different algorithms to compute the SVD. Results from studies using Subspace Iteration were better than those of Householder followed by Givens, but were considerably worse than those using Lanczos methods. Results from a few different runs using the basic version of the Lanczos algorithm are given below in Table 2. The CPU times for computing only small numbers of singular values for small matrices using Lanczos are not competitive with Householder folowed by Givens, because of the dominance of data loading times and the large overhead associated with Lanczos for data bookkeeping. Results from follow-up studies using the same data with the more specialized Lanczos algorithms:

- full reorthogonalization (FRO) [Parlett 1998],
- selective orthogonalization (SO) [Demmel 1997],
- Scott's orthogonalization (SCO) [Parlett 1998], and

- selective orthogonalization II (SO2) [Demmel 1997]

are given in Figures 12–14. Details of the methods and our implementations are documented in in [King, Kobayashi 1999].

Table 2: Lanczos Tridiagonalization and Sturm Sequencing

matrix size	no. of eigenvalues	computation time
350	5	200
350	10	300
23900	50	28700
23900	75	22800
23900	100	31200
126750	10	12700
126750	50	169900
126750	100	389800
126750	250	305200

4.5 Future Directions for Research

We can use our current SVD package as a springboard for many new directions for research. The most (seemingly) straightforward would be to extend the program to handle matrices of even larger order (10 million-by-10 million). Completion of the task requires more than just straightforward enlargement of memory space and data structures. Fast matrix and vector access and computation will require new algorithms and (possibly) different data structures. Implementation of solutions which recognize compromises, such as finding *most* of the relevant documents, rather than *all* may lead to significant speed-up in IR. If the large-scale SVD computation and IR problem are tackled, development and implementation of efficient algorithms for automatically adjusting the data structures and ranking of retrieved results with respect to the Web page database size will be required for the convenience of general users.

In our work, we noticed that input and output of data might be awkward for inexperienced users since the volume of data involved in access and retrieval is enormous. Development of computationally inexpensive and more friendly user interfaces is an important area for investigation. Specialized tools developed by computer graphics experts for visualization are often too computationally intense to be useful and are sometimes awkward for non-technical users to manipulate. Investigation of speech-based user interfaces (both isolated or combined with visual interfaces) may also lead to fruitful solutions.

Our SVD package was successfully ported to a Web-based IR system which handles both Japanese and English text data. Investigation of SVD-based ranking and retrieval from multilingual (more than two languages) databases is important if our tool will be applied to a world-wide network.

Figure 11: CPU time to compute SVD using Householder reflections and Givens rotations

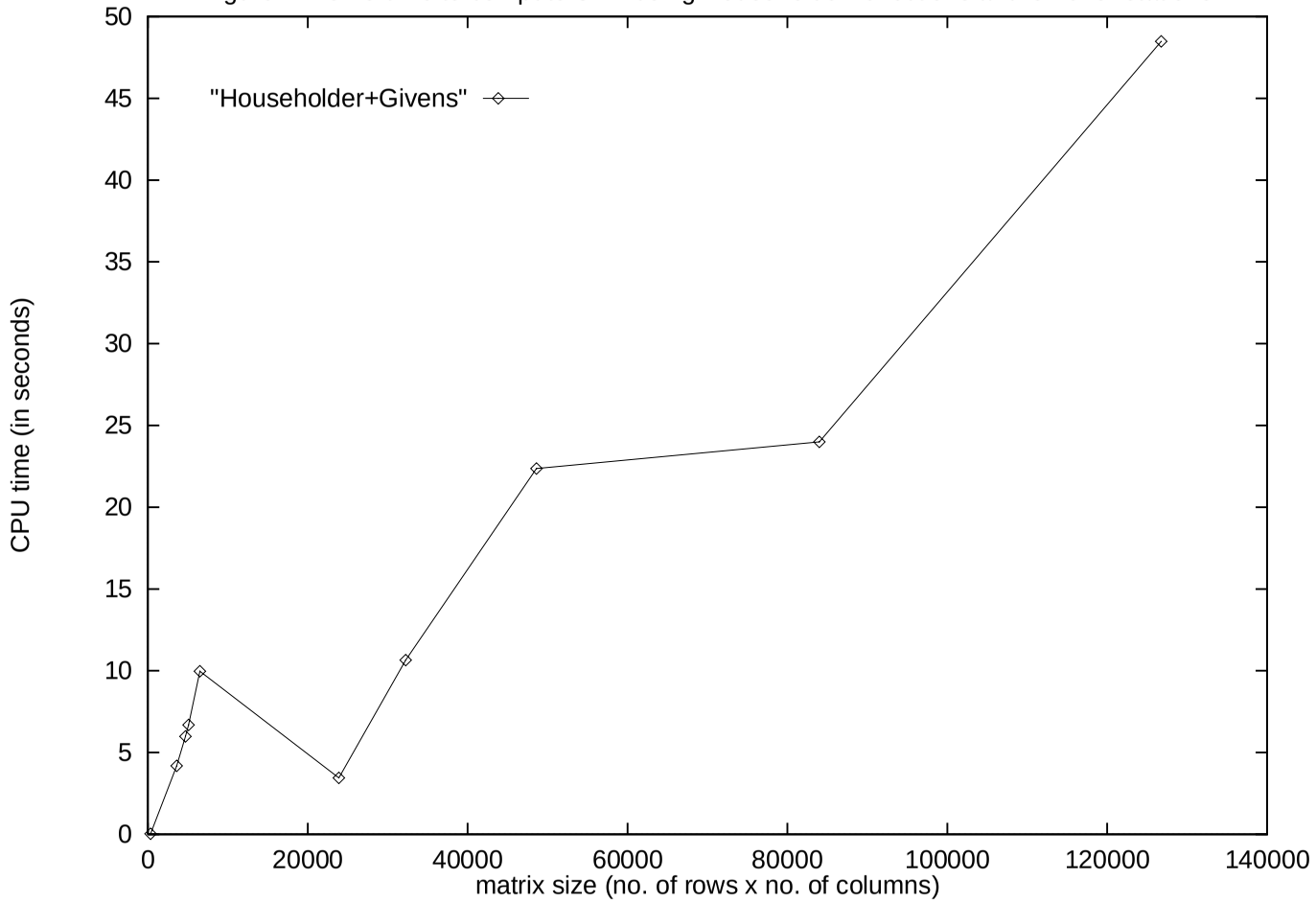


Figure 12: Lanczos Algorithm Benchmarks (8342 x 10000 matrix, 177411 nonzero entries)

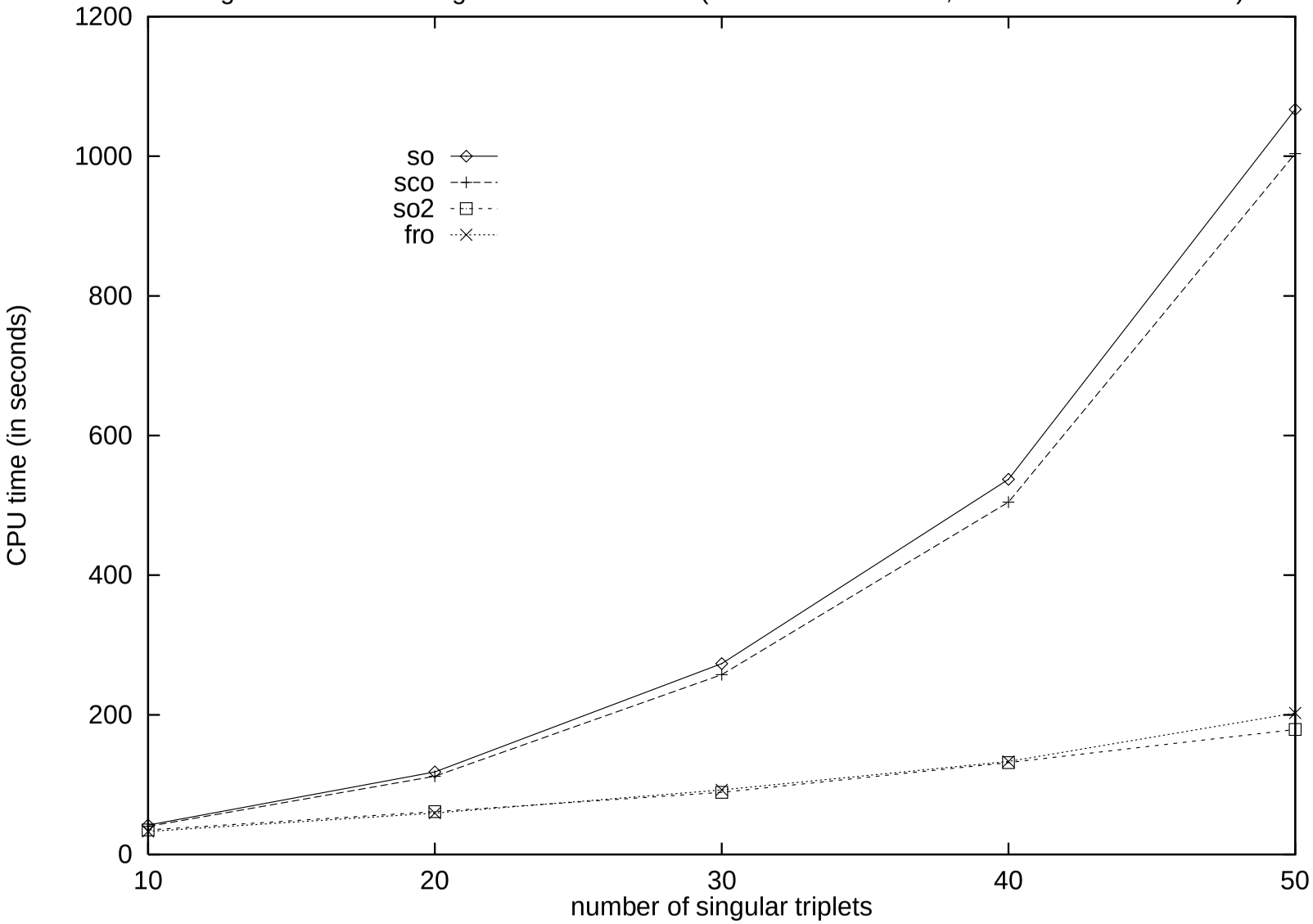


Figure 13: Lanczos Algorithm Benchmarks (1173 x 991 matrix, 3210 nonzero entries)

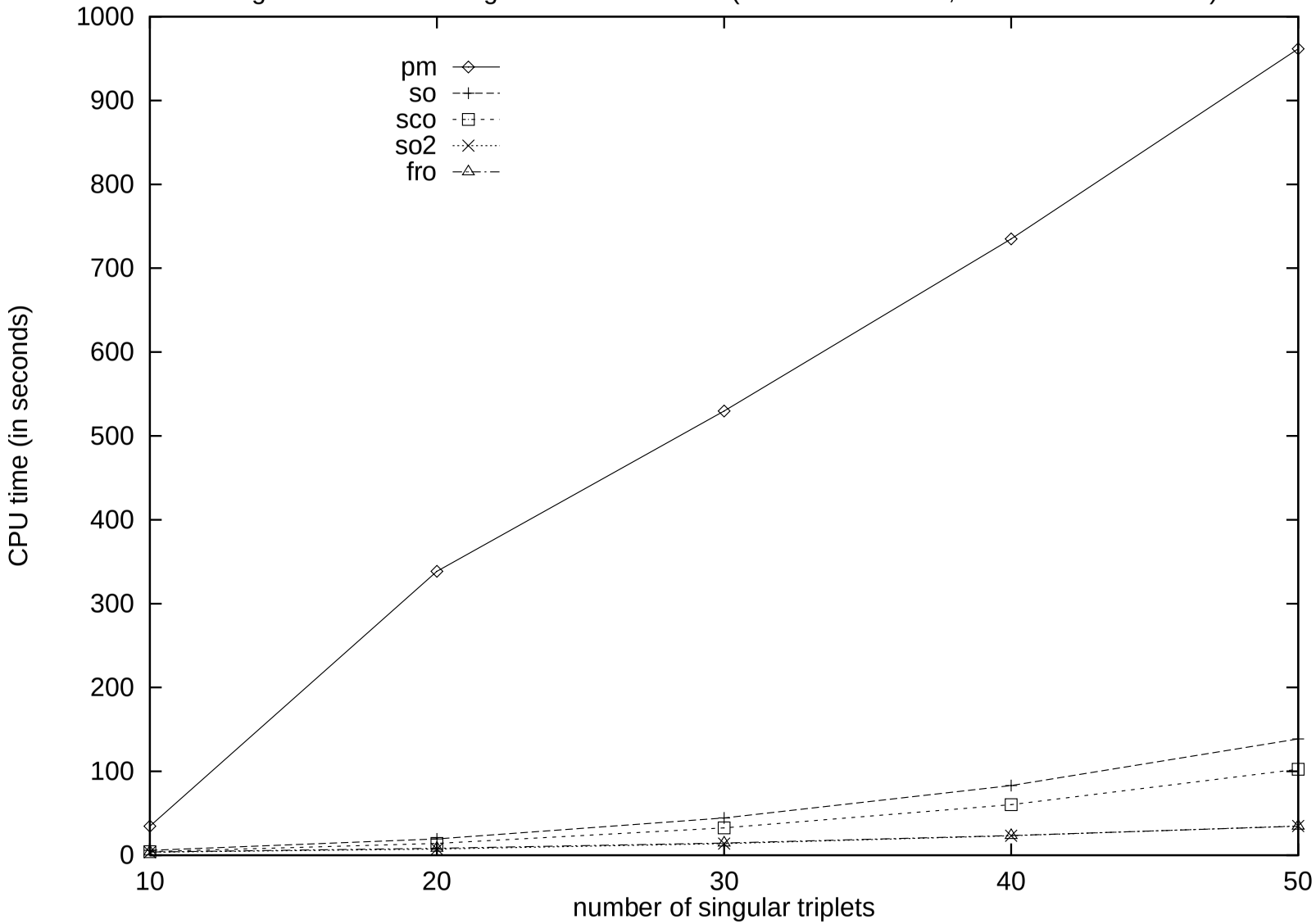
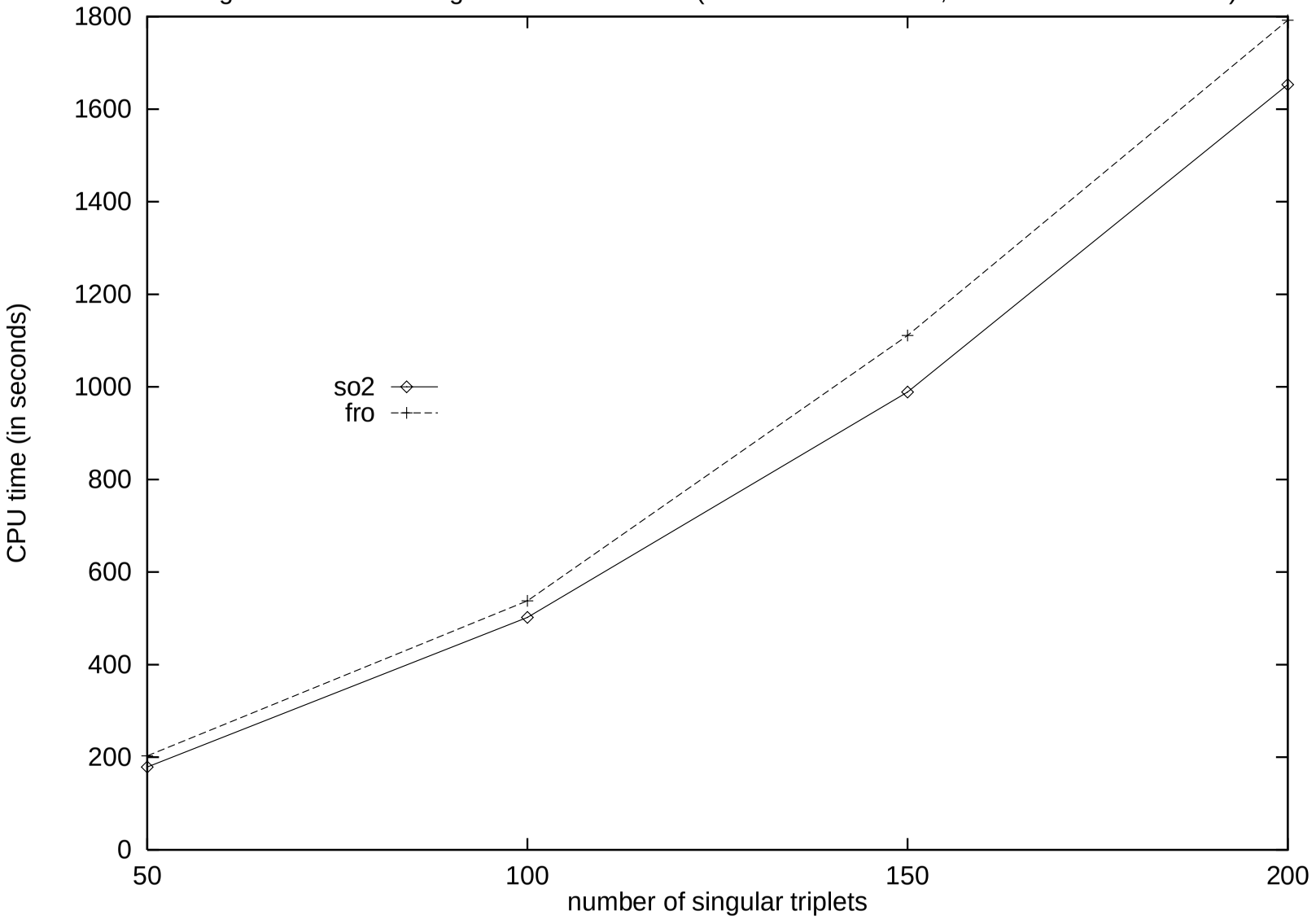


Figure 14: Lanczos Algorithm Benchmarks (8342 x 10000 matrix, 177411 nonzero entries)



Acknowledgments

This work was conducted while Georges E. Dupret was a graduate student intern at IBM Research, Tokyo Research Laboratory. The authors acknowledge helpful conversations with Hikaru Samukawa, Oliver King, Koichi Takeda, and members in the Solution Research Center of IBM Tokyo Research Laboratory.

References

[Anderson et al. 1995] Anderson, E., Bai, Z., Bischof, C., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A., Ostrouchov, S., Sorenson, D., *LAPACK Users' Guide*, second ed., SIAM, Phildelphia, PA, 1995.

[Berry et al. 1993] Berry, M., et al., " SVDPACKC: Ver. 1.0, Users' Guide", *Technical Report*, Dept. Computer Science, Univ. of Tennessee, No. CS-93-194, Oct. 1993.

[Berry et al. 1995a] Berry, M., Dumais, S., O'Brien, G., "Using linear algebra for intelligent information retrieval", *SIAM Review*, 37, 4, Dec. 1995, pp. 573–595.

[Berry et al. 1995b] Berry, M., Dumais, S., Shippy, A., "A case study of latent semantic indexing", *Technical Report*, Dept. Computer Science, Univ. of Tennessee, No. CS-95-271, Jan. 1995.

[Berry, Fierro 1996] Berry, M., Fierro, R., "Low-rank orthogonal decompositions for information retrieval applications", *Numerical Linear Algebra with Applications*, 1, 1, 1996, pp. 1–27.

[Blackford et al. 1997] Blackford, L., Choi, J., Cleary, A., D'Azevedo, E., Demmel, J., Dhillon, I., Dongarra, J., Hammarling, S., Henry, G., Petitet, A., Stanley, K., Walker, D., Whaley, R., *ScaLAPACK Users' Guide*, SIAM, Philadelphia, PA, 1997.

[Borko, Bernick 1963] Borko, H., Bernick, M., "Automatic document classification", *Journal of the ACM*, 10, 1963, pp. 151–162.

[Coleman, Van Loan 1988] Coleman, T., Van Loan, C., *Handbook for Matrix Computations*, SIAM, Philadelphia, PA, 1988.

[Conte, de Boor 1980] Conte, S., de Boor, C., *Elementary Numerical Analysis*, third ed., McGraw-Hill, NY, 1980.

[Cranor, LaMacchia 1998] Cranor, L., LaMacchia, B., "Spam!", *Communications of the ACM*, 41, 8, Aug. 1998, pp. 74–83.

[Cullum, Willoughby 1985] Cullum, J., Willoughby, R., *Lanczos Algorithms for Large Symmetric Eigenvalue Computations*, Vol. 1: Theory, Vol. 2: Programs, Birkhäuser, Boston, MA, 1985.

[Deerwester et al. 1988] Deerwester, S., Dumais, S., Furnas, G., Harshman, R., Landauer, T., Lochbaum, K., Streeter, L., "Computer information retrieval using latent semantic structure", *U.S. Patent*, No. 4839853, filed Sept. 15, 1988, issued June 13, 1989.

[Deerwester et al. 1990] Deerwester, S., Dumais, S., Furnas, G., Landauer, T., Harshman, R., "Indexing by latent semantic analysis", *Journal of the American Society for Information Science*, 41, 6, 1990, pp. 391–407.

[Demmel 1997] Demmel, J., *Applied Numerical Linear Algebra*, SIAM, Philadelphia, PA, 1997.

[Dongarra et al. 1979] Dongarra, J. et al., *LINPACK Users' Guide*, SIAM, Philadelphia, PA, 1979.

[Dumais 1991] Dumais, S., "Improving the retrieval of information from external sources", *Behavior Research Methods, Instruments and Computers*, 23, 2, 1991, pp. 229–236.

[Eckart, Young 1939] Eckart, C., Young, G., "A principal axis transformation for non-Hermitian matrices", *Bull. Amer. Math. Soc.*, 45, 1939, pp. 118–121.

[Flynn 1996] Flynn, L., "Desperately seeking surfers; Web programmers try to alter search engines' results", *New York Times*, Nov. 11, 1996, C5.

[Fox 1992] Fox, C., "Lexical analysis and stoplists" in Frakes, W., Baeza-Yates, R. (eds.), *Information Retreival*, Prentice-Hall, Englewood Cliffs, NJ, 1992, pp. 102–130.

[Golub, Kahan 1969] Golub, G., Kahan, W., "Calculating the singular values and pseudo-inverse of a matrix", *SIAM Journal on Numerical Analysis*, Ser. B 2, 1969, pp. 205–224.

[Golub, Van Loan 1996] Golub, G., Van Loan, C., *Matrix Computations*, third ed., John Hopkins Univ. Press, Baltimore, MD, 1996.

[GVU] Graphics, Visualization, and Usability Center, Georgia Institute of Technology, "GVU's WWW User Surveys": http://www.gvu.gatech.edu/user_surveys/

[Jardin, van Rijsbergen 1971] Jardin, N., van Rijsbergen, C., "The use of hierarchic clustering in information retrieval", *Information Storage and Retrieval*, 7, 1971, pp. 217–240.

[Jennings, McKeown 1979] Jennings, A., McKeown, J., *Matrix Computation*, second edition, John Wiley and Sons, NY, 1979.

[King, Kobayashi 1999] King, O., Kobayashi, M., "Information retrieval and ranking on the Web: benchmarking studies II", *IBM TRL Research Report*, RT-0298, 1999.

[Kobayashi et al. 1999] Kobayashi, M., Takeda, K., King, O., Dupret, G., "Multi-perspective retrieval, ranking and visualization of Web data", submitted to *ACM Digital Libraries '99*.

[Kobayashi, Takeda 1998] Kobayashi, M., Takeda, K., "Information retrieval on the Web: mathematical aspects", submitted to *ACM Computing Surveys*, earlier version of paper available as: *IBM TRL Research Report*, RT-0264, 1998.

[Korfhage 1997] Korfhage, R., *Information Storage and Retrieval*, John Wiley and Sons, NY, 1997.

[LANSO] *LANSO*, Dept. of Computer Science and the Industrial Liason Office, Univ. of Calif., Berkeley.

[Lawson, Hanson, 1974] Lawson, C., Hanson, R., *Solving Least Squares Problems*, Prenctice-Hall, Englewood Cliff, NJ, 1974 (available again through SIAM, Philadelphia, 1995).

[Lawrence, Giles 1998] Lawrence, S., Giles, C., "Searching the World Wide Web", *Science*, 280, April 3, 1998, pp. 98–100.

[Letsche, Berry 1997] Letsche, T., Berry, M., "Large-scale information retrieval with latent semantic indexing", submitted to *Information Sciences – Applications*, 1997.

[Liberatore 1998] Liberatore, K., "Getting to the source: is it real or spam, Ma' am ?", *Macworld Online*, Aug. 15, 1998: http://macworld.zdnet.com/features/pov.4.4.html

[Morohashi et al. 1995] Morohashi, M., Takeda, K., Nomiyama, H., Maruyama, H., "Information outlining: - filling the gap between visualization and navigation in digital libraries", *Proc. of Int'l. Symp. on Digital Libraries*, Tsukuba, Japan, Aug. 22–25, 1995,the University of Library and Information Science, pp. 151–158.

[Ossorio 1966] Ossorio, P., "Classification space: a multivariate procedure for automatic document indexing and retrieval", *Multivariate Behavior Research*, 1966, pp. 479–524.

[Parlett 1998] Parlett, B., *The Symmetric Eigenvalue Problem*, SIAM, Philadelphia, PA, 1998.

[Parlett, Dhillon 1996] Parlett, B., Dhillon, I., "Fernando's solution to Wilkinson's problem: an application of double factorization", *Linear Algebra and its Applications*, 267, 1997, pp. 247–279.

[Parlett, Scott 1979] Parlett, B., Scott, D., "The Lanczos algorithm with selective orthogonalization", *Math. Comp.*, Vol. 33, 1979, pp. 217-238.

[Press et al. 1982] Press, W., Teukolsky, S., Vetterling, W., Flannery, B., *Numerical Recipes in C*, 2nd ed., Cambridge Univ. Press, NY, 1982.

[Salton, McGill 1983] Salton, G., McGill, M., *Introduction to Modern Information Retrieval*, McGraw-Hill, NY, 1983.

[Stewart 1992] Stewart, G., "On the early history of the singular value decomposition", *Technical Report*, Dept. of Computer Science, Univ. of Maryland, College Park, No. TR-92-31, TR-2855, 1992,

anonymous ftp: thales.cs.umd.edu, directory pub/reports
Also in *SIAM Review*, 35, pp. 551–566.

[Takeda, Nomiyama 1997] Takeda, K., Nomiyama, H., "Information outlining and site outlining", *Proc. of Int'l. Symp. on Research, Development and Practice in Digital Libraries* (Tsukuba, Japan, Nov. 18–21, 1997), the University of Library and Information Science, pp. 99–106.

[van Rijsbergen 1979] van Rijsbergen, C., *Information Retrieval*, second ed., Butterworths, Boston, 1979.

[Voorhees 1995] Voorhees, E., "The cluster hypothesis revisited", *Proc. ACM Special Interest Group on Information Retreival (SIGIR)*, ACM Press, NY, 1995, pp. 188–196.

[Watkins 1991] Watkins, D, *Fundamentals of Matrix Computations*, John Wiley and Sons, NY, 1991.

[Wilkinson 1965] Wilkinson, J., *The Algebraic Eigenvalue Problem*, Clarendon Press, Oxford, U.K., 1965.