On the Enhancements of a Sparse Matrix Information Retrieval Approach

Nazli Goharian
Illinois Institute of Technology
Chicago, Illinois
nazli@ir.iit.edu

Tarek El-Ghazawi
George Mason University
Fairfax, Virginia
tarek@gmu.edu

David Grossman
Illinois Institute of Technology
Chicago, Illinois
dagr@ir.iit.edu

Abdur Chowdhury
Illinois Institute of Technology
Chicago, Illinois
abdur@ir.iit.edu

Abstract

A novel approach to information retrieval is proposed and evaluated. By representing an inverted index as a sparse matrix, matrix-vector multiplication algorithms can be used to query the index. As many parallel sparse matrix multiplication algorithms exist, such an information retrieval approach lends itself to parallelism. This enables us to attack the problem of parallel information retrieval, which has resisted good scalability. We evaluate our proposed approach using several document collections from within the commonly used NIST TREC corpus. Our results indicate that our approach saves approximately 30% of the total storage requirements for the inverted index. Additionally, to improve accuracy, we develop a novel matrix based, relevance feedback technique as well as a proximity search algorithm.

1 Introduction

With constantly growing text resources, efficiency improvements via parallel processing, storage space reduction and the improvement of search effectiveness are the main focus of information retrieval (IR) researchers. The two measures used to evaluate IR algorithms are efficiency and effectiveness. There are many ways to improve accuracy, two of which are Proximity Search [Goldman et al, 1998] and Relevance Feedback [Rocchio, 1971, Mitra et al, 1998, Chang et al, 1999]. Proximity search improves the result of the query processing by creating the capability to search for phrases, or terms in a specific window size. An example is the implementation of the Proximity Search on the King James Bible by the University of Michigan. The system searches for a text that includes the search terms specified to be Near, Not Near, Followed By, Not Followed By each other within 40, 80 or 120 characters [UMich, 1997]. Relevance Feedback improves the accuracy of the query processing by using the initial search results as additions to the initial query. The Excite@Home search engine performs the Relevance Feedback. by giving the users the option to choose "Search for more documents like this one", which searches for documents that have common words with the initial retrieved document by the initial query.

We demonstrate that our approach, same as other IR approaches, takes advantage of relevance feedback and proximity search to increase the effectiveness of the search. Text searches often rely on inverted index to reduce unnecessary I/O from retrieval of non-relevant texts, hence improving the efficiency of the search.

We present the inverted index as a compressed matrix. We demonstrate that our approach improves the efficiency by reducing the storage space compare to the conventional inverted index. The additional traditional compression techniques applied on inverted index are also applicable on our proposed storage structure. For a general review of the additional compression techniques applied on inverted index, the reader is referred to [Elias, 1975, Zobel et al., 1992, Kent et al., 1992, Witten et al., 1994, Grossman, 1995, Moffat and Zobel, 1996].

1.1 Prior Work

1.1.1 Index Compression

The use of an inverted index was shown to be the processing scheme of choice due to its reduced I/O demands for an ad-hoc query [Stone, 1987]. The storage space for inverted index structure, however, is not necessarily

```
for (count=0; count<M; count++)
        temp=0;
        for (row_ind=row_vector[count];row_ind<=(row_vector[count+1]-1); row_ind++)
                col_ind = col_vector[ind];
                temp = temp + non_zero_vector[row_ind] * Q[col_ind];
        endfor
        ScalarTPACK_output[count] = temp;
endfor
```

Figure-1: Scalar ITPACK Sparse Matrix Multiplication Algorithm

smaller than the original text. Thus, many different compression techniques are used to compress the inverted index. These include: fixed-length Byte-Aligned index compression [Grossman, 1995], variable-length compression [Witten et al., 1994, Moffat and Zobel, 1996, and Ellias, 1975], and run-length encoding techniques. Byte-aligned reduced the storage space to 15% of the size of Inverted Index. Moffat and Zobel achieved a ratio of less than 10% of the indexed text to store the index.

1.1.2 Parallel IR

Another concern with the development of scalable information retrieval is the design of algorithms that yield acceptable processing speeds when faced with large data sets. The traditional approach of parallel processing to support such functionality has yielded limited results in the information retrieval context. Prior parallel information retrieval efforts predominantly relied on special purpose techniques [Bailey, et al, 1996, Couvreur, et al, 1994, Efraimidis et al., 1995, Harabagiu et al., 1996, Lee, 1995, Lee et al., 1990, Ruocco et al., 1997, Stanfill et al., 1986, Skillicorn, 1995]. Hence, they required significant development efforts. However, many parallel sparse matrix multiplication algorithms already exist. Capitalizing on this past research for the information retrieval domain, without requiring redesign, is clearly advantageous.

1.1.3 Using Sparse Matrices for IR

The motivation of our work is to utilize other techniques and codes to implement a scalable IR system. Thus, minimizing the need for the redevelopment of software. In our recent paper [Goharian et al., 1999], we introduced a sparse-matrix information retrieval approach, which relies on the Vector Space Model to compute relevance. We showed the sparse matrix storage method as an alternative to store the inverted index and showed how to map the documents into a matrix. Two compression techniques to compress sparse matrices were utilized, namely, Horowitz [Horowitz, 1983 and Park, 1992], and Scalar ITPACK [Peters, 1991 and Petition, 1993]. Recently, our experiments demonstrated that the Horowitz approach saves very minimal storage space in comparison to conventional inverted index storage structures. Thus, we now limit our experiments to the Scalar ITPACK storage structure. Furthermore, we described and demonstrated the query processing and relevance ranking using sparse matrix-vector multiplication algorithm. Figure-1 is the algorithm for Scalar ITPACK multiplication, which is one of the commonly used sparse matrix multiplication algorithms.

Briefly reviewing the algorithm, the following steps map the document vectors to a matrix and perform the query processing. For additional details, see [Goharian, et al., 1999].

Step 1: Parse the collection to identify the unique terms of each document along with their term frequency (*tf*).

Step 2: Identify all *n* unique terms in the collection, along with documents having each term. The document frequency (*df*) and inverse document frequency (idf) are as well identified. The *idf* is commonly defined as $log(d/df)$ where *d* is the number of documents in the collection and *df* is the number of documents in the collection having the given term.

Step 3: Create document vectors and query vector with *n* dimension. The elements of the vector are either binary values of 0 or 1 for absence or presence of the term in the document or query, or a function of the *tf* and *idf* values corresponding to each term. In the vector space model, all documents and queries are represented as n-dimensional vectors [Salton, et al., 1975].

Step 4: Map the document vectors to a matrix. Each document is a row of the matrix. The columns of the matrix correspond to the unique terms in the collection. Thus, the document matrix is *MxN* dimension, where *M* is the number of documents in the collection and *N* is the number of unique terms in the collection. The nature of the text collection results in a very sparse matrix with many elements of the matrix having the value of zero where the term is absent in the document.

Step 5: Compress the sparse matrix using one of the sparse matrix compression methods.

Step 6: Perform the query processing by using one of the matrix-vector multiplication algorithms, such as Scalar ITPACK to obtain the relevance ranking.

As an aside, since every row of the matrix is independent of each other, one needs only to maintain the information associated with the current matrix row and the vector in memory at a given point in time. Clearly to reduce I/O wait time, it is necessary to fetch the information related to successive rows prior to them being needed. Note that the above description is only intended to provide a logical overview of the approach. In practice, there is no need to create the original matrix.

1.1.4 Example

The sample collection in figure-2 with five documents D0, D1, D2, D3, D4 and query Q is given.

D0 = security security social social
D1 = social security social security
D2 = social welfare system
D3 = security system
D4 = information system
Q = social security

Figure-2: Sample collection and query

Table-1 shows the unique terms for each document along with the term frequency (*tf*). The unique terms in the collection with the document frequency (*df*) and inverse document frequency (*idf*) of terms are shown in table-2.

Table-1: Term Frequency for Sample Collection

DOCS	Tf
D0	
Security	2
Social	2
D1	
Social	2
Security	2
D2	
Social	1
Welfare	1
System	1
D3	
Security	1
System	1
D4	
Information	1
System	1

Table-2: Df and Idf for Sample Collection

Term_id	Term	Df	Idf
0	Security	3	0.22
1	Social	3	0.22
2	Welfare	1	0.70
3	System	3	0.22
4	Information	1	0.70

The sparse matrix A, which represents the sample collection, is shown in figure-3. Figure-3 is the result of step 3 and step 4, described earlier. Each row corresponds to a document, i.e. document D0, D1, D2, D3 and D4. The columns correspond to the unique terms, i.e. term id 0, 1, 2, 3, and 4 in the collection. The elements of the matrix are the *tf*idf* of each term.

$$
\begin{array}{rccccc}
\text{Term id} & 0 & 1 & 2 & 3 & 4 \\
 & 0.44 & 0.44 & 0 & 0 & 0 \\
M = & 0.44 & 0.44 & 0 & 0 & 0 \\
 & 0 & 0.22 & 0.70 & 0.22 & 0 \\
 & 0.22 & 0 & 0 & 0.22 & 0 \\
 & 0 & 0 & 0 & 0.22 & 0.70
\end{array}
$$

Figure-3: Sparse Matrix A

non_zero_vector =	<0.44	0.44	0.44	0.44	0.22	0.70	0.22	0.22	0.22	0.22	0.70>
col_vector =	< 0	1	0	1	1	2	3	0	3	3	4 >
row_vector =	< 0	2	4	7	9	11>					

Figure-4: Compressed Matrix A for Sample Collection based on Modified Scalar ITPACK Compression

Q= <	0.22	0.22	0	0	0	>

Figure-5: Query Vector

```
DOC[0] = 0+(0.44*0.22)=0.10
DOC[0] = 0.10+(0.44*0.22)=0.20

DOC[1] = 0+(0.44*0.22)=0.10
DOC[1] = 0.10+(0.44*0.22)=0.20

DOC[2] = 0+(0.22*0.22)=0.05
DOC[2] = 0.05+(0.70*0)=0.05
DOC[2] = 0.05+(0.22*0)=0.05

DOC[3] = 0+(0.22*0.22)=0.05
DOC[3] = 0.05+(0.22*0)=0.05

DOC[4] = 0+(0.22*0)=0
DOC[4] = 0+(0.70*0)=0
```

Figure-6: Result of the relevance ranking

Using the Scalar ITPACK compression method, matrix *A* is compressed to three vectors, shown in figure-4. The first vector, *non_zero_vector*, indicates the non-zero elements of matrix *A*. The second vector, *col_vector*, is the column indices of non-zero elements in matrix *A*. The third vector, *row_vector*, has M+1 elements, which identifies documents *D0, D1, D2, D3 and D4*. The distance between every two adjacent elements in the *row_vector* determines the elements in the *non_zero_vector* and *col_vector* that belong to a document identified by the position of the element in the *row_vector*.

For example, to find all the terms in document D2, the distance between the values stored in positions two and three in the *row_vector*, namely 4 and 7 is computed. The three elements in *non_zero_vector* and *col_vector*, starting with the position 4, i.e. 0.22, 0.70 and 0.22 with column indices of 1, 2 and 3 belong to the second document, D2.

The query vector Q is shown in figure-5. The size of the query is for the number of distinct terms. Figure-6 shows the result of the query processing described in step 6 by using Scalar ITPACK algorithm on query Q and the sample collection. Document D0 and D1 are ranked the highest with the relevance of 0.20. Documents D2 and D3 are ranked lower, 0.05. Document D4 with the rank 0 is non-relevant to the query.

2 Efficiency

The storage space for the conventional inverted index has two components. The Index component stores the unique terms in the collection, each pointing to the Posting List. The Posting list is the list of all documents having a given term. The storage space for the Posting List is computed by considering 10 bytes for each Posting List entry, from which 4 bytes is for the document identifier, 2 bytes for term frequency and another 4 bytes for pointer to the next element in list [Grossman and Frieder, 1998].

The storage space for the compressed sparse matrix using Scalar ITPACK is computed by allocating 2 bytes per non-zero element in the first vector, which stores the tf*idf of the term, 4 bytes for each element of the second vector that is the column indices, and 4 bytes per number of documents + 1 for the third row, which is the row indices. In the case that no *tf*idf* weighting is used for computing the relevance ranking, the binary values of “1” do not need to be stored in the first row, which minimizes the storage space even further.

Table-3 shows the storage space for the conventional Inverted Index and compressed Sparse Matrix using Scalar ITPACK method for TREC 6-8 collection on disk 4-5, LA (LA Times), FT (Financial Times), FR (Federal Registry) and FBIS sub-collections, which are news documents. .

Table-3: TREC Data Storage (Bytes) Using Inverted Index and Scalar ITPACK Structures

Collection	Total Docs	Posting list Enteries	Distinct Terms	Inverted Index	S. ITPACK	Avr Ent/Term	Ratio
TREC6-8	528023	120407310	1023542	1218402688	730697208	**118**	**59.97%**
LA	131896	30200001	321087	306495228	183654116	**94**	**59.92%**
FT	210158	44571084	382437	451064958	270561762	**117**	**59.98%**
FR	55630	13685852	302943	141099722	84155294	**45**	**59.64%**
FB	130471	27444674	318767	278909478	167102534	**86**	**59.91%**

Table-4: Contrived Document Storage Space(Bytes) Using Inverted Index and Scalar ITPACK Structures

Collection	Total Docs	Posting list Enteries	Distinct Terms	Inverted Index	S. ITPACK	Avr Ent/Term	Ratio
Publications	528023	2640115000	382437	26406504118	15845096718	**6903.4**	**60.00%**
Email	1000000	5000000	500	50007000	34003004	**10000**	**68.00%**

The number of documents in each sub-collection along with the number of distinct terms in each sub-collection, excluding the stop terms, are listed under *Total Docs* and *Distinct Terms* columns of the table-3. The *Posting List Entries* column of the table shows the number of total terms in each sub-collection. The columns *Inverted Index* and *S. ITPACK* show the number of bytes used to store the indexed sub-collections. The storage space for each storage structure is calculated as described earlier. The average number of the occurrences of each unique term in each sub-collection is calculated by dividing the total number of terms by the number of distinct terms in each sub-collection. This information is listed under column *Avr Ent/Term* of table-3.

The average number of the occurrences of a term in each of the sub-collections, *Avr Ent/Term*, of the TREC data indicates that the term occurrences are relatively low. Nevertheless, the ratio of the Scalar ITPACK storage space to the conventional Inverted Index storage space is about 40% for the TREC data.

Table-4 presents the same information for the domain specific documents such as publications and email with some “believed characteristics”. The nature of the domain specific documents is such that the number of vocabularies is small. However, the terms occur more frequently in each document. For example each of the million emails in the collection, listed in table-4, has an average of 5 terms, excluding any stop term. As the result, the total number of terms, excluding the stop terms, is 5 million with 500 distinct terms in the entire email collection. This implies that each of the 500 unique terms in the Email collection occurs 10000 times. The ratio of the Scalar ITPACK storage space to the conventional Inverted Index storage space is about 40% for publications and 30% for email documents.

3 Proximity Search using Sparse Matrix Multiplication

Proximity searches are used in the Information Retrieval to increase the accuracy of the search by considering a particular query term sequence in the document. The documents, in which the query terms appear within a specific window size, are retrieved and ranked higher than any document that simply contains the query term. For example, the query “information retrieval”, with query condition of window size 1, does not rank as high the documents that have the terms “information” and “retrieval” in a sequence with a negative window size such as “Retrieval of Information”, or a window size bigger than 1 such as “Information System for Retrieval of Employee Data”.

We modified the Sparse Matrix storage structure to implement the Proximity Search. The structure and the algorithm are described using the sample collection of figure-2 with documents D0, D1, D2, D3 and D4 and query Q. Table-5 is the information provided in table-1, along with the position of each term in each document (offset).

We modified the compressed representation of matrix showed in figure-4 to implement the proximity search. We add a fourth and fifth vector, namely, *offset_vector* and *offset_marker* to the structure. Figure-7 shows the modified structure. The *offset_vector* contains the offset of each term in each given document. The number of elements in *offset_vector* is total number of non-stop terms in the collection. The elements in the *offset_marker* vector indicate the number of the occurrences of each term in a document, hence it shows which offsets in the *offset_vector* belong to a given term in a document. The number of elements in the *offset_marker* is the number of non-zero elements+1. The position of each element in the *offset_marker* corresponds to the position of the term id of the term in *col_vector*, whose offsets are identified.

For example, the difference between the values stored in the position 0 and 1 in the *offset_marker* vector, namely 0 and 2 shows that two elements of the *offset_vector*, starting in position 0, namely 0 and 1 are the offsets of term, stored in the position 0 of *col_vector*, i.e. term id 0. The difference between the values stored in the position 1 and 2 in the *offset_marker* vector, namely 2 and 4 shows that two elements of *offset_vector*, starting in position 2, namely 2 and 3 are the offsets of term, stored in the position 1 of *col_vector*, i.e. term id 1. The difference between the values stored in the position 2 and 3 in the *offset_marker* vector, namely 4 and 6 shows that two elements of *offset_vector*, starting in position 4, namely 1 and 3 are the offsets of term, stored in the position 2 of *col_vector*, i.e. term id 0.

Table-5: tf and term offset for sample collection

DOCS	Tf	Offset
D0		
Security	2	0,1
Social	2	2,3
D1		
Social	2	0,2
Security	2	1,3
D2		
Social	1	0
Welfare	1	1
System	1	2
D3		
Security	1	0
System	1	1
D4		
Information	1	0
System	1	1

The query vector is modified by adding a second vector to store the offset of the terms in the query, as shown in figure-8.

We show in figure-9 the algorithm to implement the proximity search on Sparse Matrix application of Information Retrieval.

As showed earlier in figure-6, both documents D0 and D1 are ranked the highest as the result of the query processing. The algorithm of figure-9 is applied to documents D0 and D1. From the element 0 of the *row_vector* both terms belonging to document D0 with term id 0 and 1 in the positions 0 and 1 of *col_vector* are identified. The position 0 in *offset_marker* introduces 2 elements of the *offset_vector* in positions 0 and 1, namely term offsets 0 and 1 for the term id 0. The position 1 in *offset_marker* introduces 2 elements of the *offset_vector* in positions 2 and 3, namely term offsets 2 and 3

```
non_zero_vector = <0.44   0.44   0.44   0.44   0.22   0.70   0.22   0.22   0.22   0.22   0.70>
col_vector  =         < 0      1      0      1      1      2      3      0      3      3      4 >
row_vector =          < 0      2      4      7      9      11>
offset _vector =      < 0  1  2  3  1  3  0  2  0  1  2  0  1  1  0>
offset  marker=     < 0  2  4  6  8  9  10  11  12  13  14  15>
```

Figure-7: Modified Compressed Matrix A for Proximity Search

```
Q:  v1=<0.22     0.22     0      0      0 >
    v2=<1        0        0      0      0 >
```

Figure-8: Modified Sample Query Q for Proximity Search

```
FOR each document ranked with the highest similarity score in the query processing using matrix-
vector multiplication DO
    Find from row_vector the elements (term id) and the number of elements in col_vector
    belonging to that document and the start position in col_vector.
    IF the element (term id) matches to any query term id Then DO
        FOR each found position in col_vector DO
            Find the corresponding  element in offset_marker
            Find the corresponding elements in offset_vector
        END
        Build pairs ( in the order of query terms) between the elements found in offset_vector
        across each col_vector
        FOR each pair DO
            Compute the difference between the elements of the pair
            If the difference >=1 and <= query window size
                Then mark the pair for selection
        END
    END
END
```

Figure-9: Proximity Search Algorithm

D0: (2,0) => 0-2 = -2 (2,1) => 1-2 = -1 (3,0) => 0-3 = -3 (3,1) => 1-3 = -2	**D1:** (0,1) => 1-0 = 1 (0,3) => 3-0 = 3 (2,1) => 1-2 = -1 (2,3) => 3-2 = 1
Query w/window size of 1: (The difference must be 1) Result : none	Query w/window size of 1: (The difference must be 1) Result: - social w/offset 0 and security w/offset 1 - social w/offset 2 and security w/offset 3
Query w/window size of 3: The difference must be >= 1 and <= 3 Result : none	Query w/window size of 3: (The difference must be >= 1 and <= 3) Result: social w/offset 0 and security w/offset 1 Social w/offset 2 and security w/offset 3 Social w/offset 0 and security w/offset 3

Figure-10: Result of Query Processing using Proximity Search

for the term id 1. The offset pairs across both terms are built based on the order of the query term, i.e., (term1, term0). The identified pairs are: (2,0) , (2,1), (3,0), (3,1). Document D1 is also processed similarly. The element in the position 1 of the *row_vector* identifies that two elements of *col_vector*, starting in the second position, 0 and 1 belong to document D1. The corresponding positions in the *offset_marker* are elements in positions 2 and 3, i.e., values 4 and 6. The value 4 in *offset_marker* identifies that two elements in *offset_vector* in the positions 4 and 5, i.e., offsets 1 and 3 belong to term id 0. The value 6 in *offset_marker* identifies that two elements in *offset_vector* in the position 6 and 7, i.e., offsets 0 and 2 belong to term id 1. The identified pairs are: (0,1) , (0,3), (2,1), (2,3). The result of the proximity search for query Q on the sample collection, using the algorithm described in figure-9, is shown in figure-10.

Although the initial relevance ranking ranked both documents D0 and D1 the same, the proximity search identifies that document D1 is more relevant. The window size of the query “social security” matches to the window size of the document terms in document D1 “social security social security” and not to “security security social social“ in document D0. Both query terms, “social” with term id 1 and “security” with term id 0 occur in both of the documents D0 and D1. The window size of these terms are measured in both documents by considering all possible pairs of the offsets of the terms "social" and "security" in each document. The pairs (0,1) and (2,3) in document D1 show that the window size for terms "social" with term id 0 and "security" with term id 1 is 1, which satisfies the query condition for window size of 1. In the case of query condition of window size 3, the pairs (0,1), (2,3) and (0,3) satisfy the condition of window size of smaller or equal to 3.

4 Relevance Feedback using Sparse Matrix Multiplication

Relevance Feedback is one of the common utilities in information retrieval that increases the accuracy of the retrieval. One of the earliest approaches is described in [Rocchio, 1971]. The query is refined by the results of the initial query. The steps of Relevance Feedback process are as follows:
Step 1: Issue the query.
Step 2: Select top *n* documents.
Step 3: Identify the higher *idf* terms in top documents.
Step 4: Add those terms to the initial query.
Step 5: Re-issue the query, and continue with step 2.

In the vector space model, relevant document vectors are added to the query. Figure-11 describes this process.

$$Q' = Q + c\,\mathrm{sum}(R)$$

Figure-11: Relevance Feedback for Vector Space Model

Q: original query vector
R: set of relevant document vectors
c: a constant to indicate the importance of R
Q': new query vector

We modified the Relevance Feedback algorithm of figure-11 to implement the Relevance Feedback using Sparse Matrix Multiplication algorithm, as shown in figure-12.

5 Conclusion

Previously, we introduced a sparse matrix approach to information retrieval. This approach represented the inverted index as a sparse matrix. The motivation for the approach was the reuse of prior mathematical efforts for a novel application, namely information retrieval. However, the approach was not evaluated until now where an evaluation of this approach demonstrated up to a 30% reduction in storage space over a conventional approach.

We also presented algorithms to improve retrieval accuracy when using the sparse matrix approach. To improve accuracy, proximity search and relevance feedback algorithms were developed. The proximity search technique relies on two additional vectors used to represent the offset of a term within the document. The relevance feedback approach mapped a traditional relevance feedback algorithm to the matrix domain.

In the future, we will evaluate the approach using traditional information retrieval measures such as precision and recall and will implement the approach on a parallel platform.

```
Step 1: Find top n relevant documents by using Matrix Multiplication algorithm.
Step 2: Find the first k term_id of terms in n documents, which are more relevant documents, with
        higher idf.
Step 3: Add these terms to query by replacing existing value 0 in the corresponding column in the
query vector with tf*idf of terms or with the binary value 1 for the existence of the term.
Step 4: Re-calculate the relevance using the Matrix Multiplication algorithm.
```

Figure 12: Relevance Feedback Algorithm using Sparse Matrix Multiplication

References

[Bailey et al., 1996] P. Bailey, and D. Hawking, A Parallel Architecture for Query Processing over A Terabyte of Text, CS Tech Report, The Australian National University, June.

[Chang et al., 1999] C. Chang, C. Hsu, Enabling Concept-Based Relevance Feedback for Information Retrieval on the WWW, IEEE Trans. on Knowledge and Data Eng., 11(4).

[Couvreur et al., 1994] T. Couvreur, R. Benzel, S. Miller, D. Zeitler, D. lee, M. Singhal, N. Shivarati, W. Wong, An Analysis of Performance and Cost Factors in Searching Large Text Databases Using Parallel Search Systems, JASIS.

[Elias, 1975] P. Elias, Universal Code word sets and representations of the integers, IEEE Trans. IT-21, 2(Mar).

[Efraimidis et al., 1995] P. Efraimidis., C. Glymidakis, B. Mamalis, P. Spairakis, and B. Tampakas, Parallel Text Retrieval on a High Performance Supercomputer Using the Vector Space Model, ACM SIGIR'95.

[Goharian et al., 1999] N. Goharian, T. El-Ghazawi, D. Grossman, On the Implementation of Information Retrieval as Sparse Matrix Application, PDPTA'99, vol. 3, pg 1551-1557.

[Goldman et al., 1998] R. Goldman, N. Shivakumar, S. Venkatasubramanian, H. Garcia-Molina, Proximity Search in Databases, VLDB'98, 26-37.

[Grossman and Frieder, 1998] D. Grossman & O. Frieder, Information Retrieval: Algorithm and Heuristics, Kluwer Academic Publishers.

[Grossman, 1995] D. Grossman, Integrating Structured Data and Text: A Relational Approach. PhD Thesis, GMU.

[Harabagiu et al., 1996] S. Harabagiu, and D. Moldovan, A Parallel Algorithm for Text Inference, IEEE IPPS'96.

[Horowitz, 1983] E. Horowitz & S. Sahni, Fundamentals of Data Structures, CS Press.

[Kent et al., 1992] A. Kent, A. Moffat, R. Sacks-Davis, R. Wilkinson & J. Zobel, Compression, Fast Indexing, and Structure Queries on a Gigabyte of Text, TREC-2.

[Lee et al., 1990] D. Lee, and F. Lochovsky, HYTERM – A Hybrid Text-Retrieval Machine for Large Databases, IEEE TC 39(1), Jan..

[Lee, 1995] D. Lee, Massive Parallelism on the Hybrid Text-Retrieval Machine, IP&M, Vol. 31(6).

[Mitra et al., 1998] M. Mitra, A. Singhal and C. Buckley, Improving Automatic Query Expansion, ACM SIGIR'98.

[Moffat and Zobel, 1996] A. Moffat and J. Zobel, Self-Indexing Inverted Files for Fast Text Retrieval, ACM TOIS. 14 (4):349-379.

[Park et al., 1992] S. Park, J. Draayer, and S. Zheng, Fast sparse matrix multiplication, Comp. Phys. Comm., Vol. 70.

[Peters, 1991] A. Peters, Sparse matrix vector multiplication technique on the IBM 3090 VP, Parallel Computing 17.

[Petition et al., 1993] S. Petition, Y. Sood, K. Wu, W. Ferng, Basic sparse matrix computations on the CM-5, J. Mod. Phys., C(1).

[Ruocco et al., 1997] A. Ruocco, and O. Frieder, Clustering and Classification of Large Document Bases in a Parallel Environment, JASIS.

[Rocchio, 1971] J.Rocchio, The SAMR Retrieval System Experiments in Automatic Document Processing, Prentice Hall.

[Salton et al., 1975] G. Salton,, C. Yang, and A. Wong, A Vector Space Model for Automatic Indexing, CACM, 18(11).

[Skillicorn, 1995] D. Skillicorn, A Generalization of Indexing for Parallel Document Search, Structured Parallel Computation in Structured Documents, Queen's University, Kingston, Ontario.

[Stanfill et al., 1986] C. Stanfill, B. Kahle, Parallel Free-Text Search on the Connection Machine System, CACM, 1200-1229

[Stone, 1987] H. Stone, Parallel querying of large databases: A case study, IEEE Computer, 20(10).

[UMich,1997] www.hti.umich.edu/relig/kjv/prox.html.

[Witten et al., 1994] I. Witten, A. Moffat, and T. Bell, Managing Gigabytes. Van Nostrand Reinhold.

[Zobel et al., 1992] J. Zobel, A. Moffat and R. Sacks-Davis, An Efficient Indexing Technique for Full Text Database Systems, 18th VLDB pg. 352-362.