An Information Retrieval Test-bed on the CM-5

Brij Masand and Craig Stanfill
Thinking Machines Corporation
245 First Street
Cambridge MA 02142

I. INTRODUCTION

For many years, research on information retrieval was mostly confined to a few relatively small test collections such as the Cranfield collection [1], the NPL Collection [2], and the CACM Collection [3]. Over the years, results on those collections accumulated, with the aim of determining which technique or combination of techniques resulted in the best precision/recall figures on those collections. Gradually, a "standard model" more-or-less emerged: for the test collections under study, consistently good results are obtained by vector-model retrieval using a cosine similarity measure, *tf.idf* weighting, and a stemming algorithm (e.g. Chapter 9 of [4], [5]).[1] Out-performing this model on the old test collections has proved extremely difficult. This has led to a danger of stagnation in the field of IR, and a feeling that the majority of what can be learned from precision-recall experiments on the old collections has been learned.

Fortunately for the field, a number of recent developments have led to new challenges for the field. Among these:

- The Tipster project [6] has led to the construction of a test collection of unprecedented size [7]. This leads to challenges related to scaling up IR methods.
- IR methods are now being used on full documents (e.g. news stories), rather than on abstracts. This leads to new challenges relating to the structure of large documents, particularly effects relating to term proximity.
- The on-line database vendors have shown interest in full-text IR. This leads to challenges relating to the integration of IR methods with traditional boolean methods which, in the operational environment, must continue to be supported if only for the benefit of the existing, entrenched user community.
- Network-oriented IR protocols have become increasingly popular as a basic tool for navigating across the network. This leads to the challenge of designing systems which give uniformly interpretable results across a distributed database.

The work reported in this paper represents some steps along the way to solving these problems. In essence, the challenge we are facing is to design a system which delivers high precision-recall figures on large databases of long, complex documents. Furthermore, the system must be compatible with existing boolean retrieval methods, and it must be possible to use this system in a distributed database in a manner which yields consistently interpretable results. The specific steps reported at this point consist of the following:

- A database architecture which is suitable for use on large collections of structured documents, which supports both base-line IR and boolean retrieval methods.
- An in-memory implementation of this architecture on a massively parallel computer (the Connection Machine model CM-5), which is used as test-bed.
- Precision-Recall figures for this test-bed applied to the Tipster collection, as part of TREC.

It must be emphasized that this work is in an early stage and, at this point, has not reached the point where these methods are demonstrable on extremely large databases. Furthermore, work relating to taking advantage of document structure has only barely begun.

II. DATABASE ARCHITECTURE

A. Choice of a Representation

There are three basic approaches to representing databases for retrieval applications:

- **Text Files**. In this method, the database is stored in full-text form, and scanned sequentially. Such methods are often implemented by special-purpose hardware [8].
- **Signature Files**. In this family of methods, a compressed surrogate for the text file is searched instead of the text file itself. Overlap encoding is usually employed to construct the surrogate. There are many variants on signature files [9].
- **Inverted Files**. In this family of methods, an index containing the locations of every word in the database is constructed. The search is then accomplished by reference to

1. Other approaches yield results which are nearly as good and, in rare cases better. Approaches based on Bayesian inference networks are particularly notable in that respect. However, the above model has come to be accepted as a de-facto standard against which new methods are measured.

the index. Again, there are many variants on inverted file representations [10].

Inverted files have proved the only technique which supports interactive access to very large databases; this is primarily due to the excessive I/O requirements of the other methods. Within the family of inverted file methods, several variants are possible:

- **Simple Inverted Files**. In a simple inverted file, the index consists of a list of documents in which each word occurs.
- **Inverted Files with Weights**. In an inverted file with weights, the above information is supplemented with weights, in support of vector retrieval methods.
- **Structured Inverted Files**. In a structured inverted file, the index captures structural information (typically the paragraph/sentence/word coordinates of each occurrence of each term), in support of boolean retrieval methods which rely on proximity operators (e.g. word adjacency).

In this research, we have decided to explore the use of structured inverted files for the following reasons:

- They support the proximity operations required as part of boolean retrieval systems. This support makes architectures based on structured inverted files more likely to be adopted by the major on-line vendors.
- They provide a fundamentally richer representation of document structure than is available with the other methods.
- They are collection-independent and retrieval-method independent.

This last point requires some explanation. In a distributed environment, it would be useful to be able to search text files at multiple locations as if they were a single text file. Once weights — which are both collection-dependent and retrieval-method dependent — are incorporated into the index, transparent distributed access becomes impossible. The product of this is that the results of a single query applied to multiple databases cannot be meaningfully combined, since there is no way to compare the ranks or scores applied to the documents returned.

One of the primary challenges associated with the use of structured inverted files is their bulk: there are approximately as many index entries in the inverted file as there are words in the database, and each entry must represent a document, paragraph, sentence, and word coordinates for that word. This problem has been substantially solved by use of a novel combination of compression techniques [11], which allow structured indexes having a bulk on the order of 1/3 the size of the full text to be constructed.

The second challenge associated with the use of structured inverted files is to implement the standard information retrieval model using them. The results of this effort are reported in this paper.

The final challenge associated with structured inverted files is using them to implement methods which go beyond the standard model, taking into account the added richness the structured representation to improve retrieval system performance for databases having long, structured documents. This final challenge remains a topic for future research, and there are no results to report at this time.

B. The Database Architecture

The database consists of the following structures:

- **A Compressed Structured Posting File**. The first component of the database is an array of compressed postings. Each posting gives the location of a word expressed as a document-paragraph-sentence-word 4-tuple. The postings are sorted by word ID, in ascending document order.
- **A Lexicon/Index**. The second component of the database is a lexicon/index. For each word in the database, it stores the number of times it occurs plus the location of its postings in the posting file. The lexicon is structured so that the information pertaining to a given word may be quickly located.
- **Document Information**. The final component of the database is an array of document-related information. Each entry contains a mapping from an internal document identifier (an integer) to an external document identifier (a string). Additional information, such as the length of the document or normalization information, may be added as necessary.

The following operations are supported:

- **Adding Postings**. A 5-tuple consisting of a word plus its four coordinates may be added to the database.
- **Extracting Postings**. A word is presented to the database. An array having the decompressed coordinates of all occurrences of that word is returned. The array is sorted by ascending document coordinate, with paragraph, sentence, and word coordinates used as additional sort keys as needed.
- **Extracting Lexicon Information**. Lexicon information such as the number of occurrences of a word may be extracted.
- **Setting Lexicon Information**. Supplemental lexicon information (e.g. part-of-speech information) may be added to the lexicon. This feature is not used in the work reported herein.
- **Defining a Document**. Defines document-specific information such as external document identifier, document location on disk, etc., and associates it with a document coordinate.
- **Extracting Document Information**. Extracts the document-specific information mentioned above, given a document coordinate.

We believe that this architecture suffices to implement boolean retrieval methods, standard IR methods, and novel methods making use of document structure, and that it can be scaled to very large databases and to massively parallel computers.

C. The Test-bed

The above architecture has been implemented in test-bed form on the Connection Machine model CM-5. The purpose of this implementation is to permit various retrieval methods to be evaluated, rather than to support on-line services on very large files. As such, the focus was on simplicity of implementation, speed of database construction, and speed of query execution, rather than on handling files larger than a few Gigabytes.

The CM-5 [12] is a general-purpose parallel computer constructed from commodity microprocessors. Each processing node consists of a SPARC microprocessor, 32 megabytes of RAM, and a network interface. The CM-5 has two user-accessible networks: a *Data Router*, which is used for point-to-point transmission of data packets, and a *Control Network* which is used to implement global operations such as barrier synchronization, broadcast, and global-maximum. The data router provides each processor with 5 MB/second (full duplex) worth of point-to-point bandwidth. The control network is constructed using a fan-in/fan-out tree so that global operations complete within a few microseconds of their initiation.

The CM-5 I/O system consists of a high-performance mass storage system called the *scalable disk array* (*SDA*). In this system, disk controllers are connected directly to the CM-5's data router. This allows all processors to have equal access to all disks in the system, providing the image of a scalable shared disk environment. The file system implements a UNIX file system on top of this hardware, such that file systems may be striped across up to 256 disks [13]. The result is a file system which can sustain transfer rates exceeding 150 MB/second in large configurations.

The basic approach taken in this implementation is to begin by partitioning the document set among the processors. This is done by having each processor read a fixed-size contiguous chunk (1 MB) of data from the input file. In general, this will result in some documents spanning processor boundaries, so document fragments will then be re-distributed. Once this is done, each processor parses and lexes its own documents, using conventional programming techniques. The postings are then inserted into the inverted file.

One novel implementation trick is used in this phase of processing: rather than sorting the postings, which would be very time consuming owing to the size of the uncompressed postings, the database is indexed in two passes. On the first pass, called the "dry run", the postings are generated, counted, and discarded. At the end of this pass, the system knows how much space is required to store the posting list for each word. Space is then allocated and carved up. The second "production" phase then begins. The database is scanned, lexed, and indexed again from scratch, but this time the postings are compressed and stored into the space allocated at the end of the dry run. This strategy doubles indexing time but, by eliminating the expense of sorting the posting file, it ends up both simplifying the software and reducing overall database construction time.

At the end of this phase, the data structures noted above (posting file, lexicon/index, and document information) have been constructed in-memory, and the database is ready for querying. Using these methods on a 64 processor CM-5, the TREC-2 training database (2.2 Gigabytes) required 20 minutes to index. The speed of the indexing software permits the database to be re-indexed for every retrieval experiment, allowing both indexing methods and query methods to be conveniently explored. Ignoring stop words, the size of the compressed inverted file index for the TREC database is 24% of the raw text. Details of the compression algorithm can be found in [11].

The first step in query evaluation is to broadcast the query to all processors. In a boolean system, the query would then be processed locally by each processor, then the results produced by the processors concatenated to return the final answer.

In an information retrieval system based on term weighting and document ranking, slightly more work is required. First, query-term weights are generally based on some sort of term-frequency observations. These cannot be done locally, but require the use of some simple global operations. For example, to determine the number of documents in which a word occurs, each processor would count document occurrences for itself, then the *global sum* operation (supported in hardware by the Control Network) would be used to produce a machine-wide count. The second problem which arises comes after the documents have been scored: an algorithm is required to extract the highest-ranking documents in the collection. Several parallel algorithms for this task have been described in [14]. The test-bed used a variant on the iterative extraction algorithm: each processor locally determines its highest-ranking documents, then repeated application of the *global maximum* operation are used to find the best documents in the collection.

III. THE COSINE SIMILARITY MEASURE

Results of using the classical cosine similarity measure on the TREC collection have already been reported elsewhere [15], so those results have not been replicated. This section will briefly describe how the cosine measure may be implemented within our architecture.

Cosine similarity measures generally involve constructing an inverted file incorporating document-term weights. The structured inverted file architecture does not provide this information, partly because it is inconsistent with the structured representation, and partly because of the difficulties noted above with regard to distributed databases. Except for the cosine measure's document normalization factor, we find that it is possible to implement document-term weighting based only in information in the structured posting file.

In general, term weights (both in queries and documents) may be computed based on the following factors [5], all of which are available at run-time in our architecture.

- **Term Frequency**. This is the number of times a term occurs in the database. This information is retained in the lexicon.
- **Document-Term Frequency**. This is the number of documents in which a term occurs. It may be computed at run-time by traversing posting list for the term, counting initial occurrences of the term.
- **In-Document Frequency**. This is the number of times a term occurs in a given document. This may be computed at run-time by traversing the posting list for the term, counting the number of occurrences of the term having the same document coordinate.
- **Maximum In-Document Frequency**. This is the maximum number of times a given term occurs in any document. It may be computed directly from the in-document frequencies.

Most conventional weighting schemes (e.g. tf.idf) may be computed from these run-time factors. The advantage of doing these computations at run-time is that it eliminates the need to incorporate collection-specific information into the database at indexing time. This is important for dynamic collections as well as for distributed databases, as described above.

The difficulty with these methods when applied to the cosine norm is that the total-document-weight term cannot be conveniently computed at run-time using an inverted file. It must, therefore, be computed at index time and stored separately (e.g. in the document-specific information data structure).

There are two solutions to this difficulty. The first is to insist on using the cosine norm, accepting the difficulties that this implies. The second is to look for alternatives to the cosine norm. We believe that this is the more promising approach, but this is in the realm of work-not-yet-completed. In any event, the value of the cosine norm for large structured documents has not been established at this time.

IV. RETRIEVAL EXPERIMENTS:

Most of the time (about 3 to 4 person months) for our TREC participation was spent on building the test bed. However we did finish some runs with automatically constructed routing and adhoc queries for which we report the results here. The experiments were done using the entire collection. We compare words and phrases, mixed case vs. lower case and also explore document length normalization and proximity measures.

A. Retrieval for Routing Topics:

All queries were constructed and optimized automatically. The terms consisted of words (ignoring stop words) as well as all phrases consisting of adjacent words. Numbers were ignored and case was preserved. No stemming or query expansion with thesauri etc. was attempted.

Routing queries were constructed by looking at each word and (adjacent) phrase from the whole text of the topic templates and determining a weight based on the number of relevant documents present in the first 100 retrieved documents, by using just that term. An initial weighted query was constructed for each topic by the above process. Then each topic query was optimized by choosing thresholds (per topic) for the weights and rejecting all weighted query terms below the threshold. The optimum threshold for each topic was chosen by straightforward incremental search. Table 1 shows the results for the routing experiments. Routing queries with both

Table 1: Routing Queries

Method	Precision at 100 docs	Average precision
tmc6- routing-words-phrases	.3396	.2553
tmc7-routing-words	.2920	.2045
tmc6-routing-words-phrases-**ip**	.3750	.2716
tmc6-routing-words-phrases-**doc-length-sent-prox**	.3782	.2792
tmc6-routing-words-phrases-**ip-sent-prox**	.3856	.3344

weighted words and phrases (queryid tmc6) did better than queries using just words (queryid tmc7). Using the same (official) queries, but adding sentence level proximity (**sent-prox**), document length scaling (**doc-length**) and inverse weights based on which paragraph the term appears in (**ip**), seem to improve results (see the next section for more details about the techniques).

B. Retrieval for Adhoc Topics

Adhoc queries were automatically constructed by using words and phrases from different sections of the topic templates and using tf.idf weights (as derived from the training collection). The "best" sections for the new topics were chosen by experimenting with the training topics. Queries derived from the description-concept sections were used for most of the experiments. A threshold for weights was used to select terms for the final queries. Table 2 shows the results for the adhoc queries.

Table 2: Adhoc Queries

Method	Precision at 100 docs	Average precision
Case:		
tmc8-adhoc-dcwp-idf-**caps-wt**	.2002	.1276
tmc8-adhoc-dcwp-idf-**lower-wt**	.1734	.1157
Document-length and inverse-para-scaling:		
tmc8-adhoc-dcwp-idf-**lower-doc-length-wt**	.3308	.1904
tmc8-adhoc-dcwp-idf-**caps-ip-wt**	.3432	.1939
tmc8-adhoc-dcwp-idf-**lower-ip-wt**	.3422	.2027
tmc9-adhoc-etwp-idf-**caps-ip-wt**	.3144	.1736
Stemming:		
tmc8-adhoc-dcwp-idf-**lower-stem-wt**	.1670	.1152
tmc8-adhoc-dcwp-idf-**lower-stem-ip-wt**	.3240	.1980
Proximity:		
tmc8-adhoc-dcwp-idf-**caps-doc-length-sent-prox-wt**	.3436	.2012
tmc8-adhoc-dcwp-idf-**lower-doc-length-sent-prox-wt**	.3518	.2146
tmc8-adhoc-dcwp-idf-**caps-ip-para-prox-wt**	.2892	.1681
tmc8-adhoc-dcwp-idf-**lower-ip-para-prox-wt**	.3006	.1772
tmc8-adhoc-dcwp--idf-**caps-ip-sent-prox-wt**	.3476	.1988
tmc8-adhoc-dcwp-idf-**lower-ip-sent-prox-wt**	.3602	.2164

The query tmc8 (dcwp) consisted of words and phrases from the description and concept sections of the topic templates. Query tmc9 (etwp) used words and adjacent phrases from the entire topic. Bold-face acronyms emphasize particular experiments with case (**caps** and **lower**), sentence and paragraph level proximity (**sent-prox** and **para-prox**) document length scaling (**doc-length**), inverse weights based on paragraph position (**ip**) and weight thresholds (**wt**). The queries were not changed for the different experiments.

Idf weighted terms from the description and concept sections taken together (query tmc8) seem to do better than those derived from the entire topic (query tmc9).

C. Case

For the adhoc queries, we compared indexing with and without preserving case (similar treatment for the queries). Except for the simplest experiment with weight thresholds, converting everything to lower case seems to yield comparable or better results than upper case. A similar experiment for routing queries wasn't attempted because that would have required reformulating and reoptimizing the routing queries.

D. Stemming

Using the Porter Algorithm software for stemming from [16] we experimented with stemming at index time (and stemming the queries). We found that stemming reduces performance when compared with similar experiments using lower case -- since the software we had used lower case. We are not sure yet why there is such a decrease.

E. Document length and Term position

Document length scaling was used to explore the effect of emphasizing shorter documents. A linear decreasing scaling for longer documents, with a tail was used. An inverse weight based on the paragraph the term appears in, was also explored. Both the document length scaling and the inverse paragraph scaling increase performance significantly and seem to be comparable to each other.

F. Proximity

The postings for the inverted file allow use of term position. Experiments are underway to define proximity scoring methods that enhance weights for terms appearing close together (clusters of terms), and can also be implemented efficiently within the current architecture. We have achieved good results with sentence level proximity measures based on a bonus score for the query terms that appear within the same sentence and within a certain distance of each other. The bonus is also proportional to the term weight itself. Experiments that used a bonus independent of term weight dramatically reduced performance (numbers not reported here), possibly due to noise introduced by clusters of unimportant terms. Similar experiments with paragraph level proximity yielded significantly poorer results as compared to sentence level proximity. Finally combining either document length or inverse paragraph scaling with sentence level proximity improved results.

V. CONCLUSIONS

We have successfully implemented an in-memory compressed inverted file text retrieval system on the CM5 Connection Machine system.

Queries that used both words and phrases composed of adjacent words did better that those that used words alone. While our experiments completed so far suggest that converting everything to lower case for adhoc queries seems somewhat better, it is not clear whether the minor differences couldn't be removed by further optimization of other parameters. Experiments that take document length and term position into account suggest that normalizing for document length and increasing the weight of terms appearing earlier in the documents lead to significant improvements for both routing and adhoc queries. We have also seen that proximity scores based on nearby terms in a sentence improve retrieval performance.

VI. FUTURE WORK

We would like to compare the effects of cosine normalization with what we have tried so far and also explore its interaction with techniques that use term position and proximity measures.

VII. REFERENCES

[1] Cleverdon, C. W., Mills, J. & Keen, E. M. (1966). *Factors Determining the Performance of Indexing Systems, Vol 1: Design, Vol. 2: Test Results*. Aslib Cranfield Research Project, Cranfield, England, 1966.

[2] Sparck Jones, K. and Webster, C.A. (1979). *Research in Relevance Weighting*, British Library Research and Development Report 5553, Computer Laboratory, University of Cambridge.

[3] Fox, E. (1983). Characteristics of Two New Experimental Collections in Computer and Information Science Containing Textual and Bibliographic Concepts. *Technical Report TR 83-561*, Cornell Univerisy: Computing Science Department.

[4] Salton, G. (1989). *Automatic Text Processing*. New York: Addison-Wesley.

[5] Salton, G & Buckley, C. (1988). Term-Weighting Approaches in Automatic Text Retrieval. *Information Processing and Management*, **24** (5), pp. 513-523.

[6] Harman, D (1993). The DARPA TIPSTER Project. *SIGIR Forum*, **26** (2), pp. 26-28.

[7] Harman, D (1993). Overview of the First TREC Conference. *Proceedings, SIGIR-93*, pp. 36-47. Pittsburgh.

[8] Hollaar, L. A. (1992). Special-Purpose Hardware for Information Retrieval. In *Information Retrieval: Data Structures and Algorithms*. Frakes, B. & Baeze-Yates, R. eds. Englewood Cliffs, New Jersey: Prentice Hall.

[9] Faloutsos, C. (1992). Signature Files. In *Information Retrieval: Data Structures and Algorithms*. Frakes, B. & Baeze-Yates, R. eds. Englewood Cliffs, New Jersey: Prentice Hall.

[10] Harman, D., Fox, E. Baeza-Yates, A. & Lee, W. (1992). Inverted Files. In *Information Retrieval: Data Structures and Algorithms*. Frakes, B. & Baeze-Yates, R. eds. Englewood Cliffs, New Jersey: Prentice Hall.

[11] Linoff, G. & Stanfill, C (1993). Compression of Indexes with Full Positional Information in Very Large Text Databases. *Proceedings, SIGIR-93*, pp. 88-95. Pittsburgh.

[12] Thinking Machines Corporation (1991). Connection Machine CM-5 Technical Summary. Cambridge, MA: Thinking Machines Corporation.

[13] Lo Verso, S., Isman, M., Nanopoulos, A., Nesheim, W., Milne, E. & Wheeler, R. (1993). SFS: A Parallel File System for the CM-5. *Proceedings, 1993 Summer USENIX Technical Conference*. pp. 291-306. Cincinnati, OH.

[14] Stanfill, C. (1992). Parallel Information Retrieval Algorithms. In *Information Retrieval: Data Structures and Algorithms*. Frakes, B. & Baeze-Yates, R. eds. Englewood Cliffs, New Jersey: Prentice Hall.

[15] Salton, G. & Buckley, C. (1992). SMART Trade-offs. *Proceedings, TREC-1*. Gaithersburgh, MD.

[16] Frakes, B.(1992). Stemming Algorithms. In *Information Retrieval: Data Structures and Algorithms*. Frakes, B. & Baeze-Yates, R. eds. Englewood Cliffs, New Jersey: Prentice Hall.