IST-Africa 2014 Conference Proceedings
Paul Cunningham and Miriam Cunningham (Eds)
IIMC International Information Management Corporation, 2014
ISBN: 978-1-905824-44-1

Data Structures for Information Retrieval

Denis L. Nkweteyim
University of Buea, P.O. Box 63, Buea, Cameroon
Tel: +237 75229038, Fax: +237 3332 2272, Email: nkweteyim.denis@ubuea.cm

Abstract: The process of efficiently indexing large document collections for information retrieval places large demands on a computer's memory and processor, and requires judicious use of these resources. In this paper, we describe our approach to constructing such an index based on the vector-space model (VSM). We review the stages involved in generating an index, for weighting the index terms, and for representing documents in the VSM. We explain our choice of data structures from the parsing of the document collection through the generation of index terms, to generation of document representations. We explain tradeoffs in our choice of data structures. We then demonstrate the approach using the OHSUMED data set. Our results show that even with only a modest amount of main memory (4 GB), large data sets such as the OHSUMED data set can be quickly indexed.

Keywords: Information retrieval, data structures, index, dictionary, posting, term frequency, vector-space model, binary search tree, linked list

1. Introduction

The term information retrieval (IR) is used to describe the process of "finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers)" [1]. Full text retrieval systems like search engines enable search and access to document content using mechanisms that go beyond document titles, author names, or other high level metadata. What makes information retrieval different from, and much more difficult than traditional relational database search has to do with the fact that the search is done on unstructured documents, i.e., documents that lack the kind of well-defined structure provided by attributes, that is inherent in relational database tables. Whereas relational database search involves searching for the presence or absence of objects that have specified values in well-defined fields, unstructured documents like text files and web pages can only be searched by examining all the tokens, usually words, which make up these documents.

The traditional approach to information retrieval requires indexing of all the documents on which the search would be performed, then at search time, when the user provides a search criterion or query (i.e., a set of terms used to describe the search), the IR system should generate a set of documents that hopefully, meet the search criterion. Indexing involves extracting the features of the document, typically words.

IR systems can be differentiated in the ways they use the query to generate relevant documents. In Boolean retrieval, every index term is represented as a Boolean vector whose size is the same as the number of documents in the collection. In this scheme, a zero term in the vector signifies absence of the term in the corresponding document and a one term, the presence of the term in the document. A query then is a Boolean expression of term vectors in which terms are combined with the operators AND, OR, and NOT. Shortcomings with Boolean retrieval are well documented (see for example, [1] [2] [3] [4]). These shortcomings include the following:

1. Ranking of retrieved documents based on their relevance is difficult, as Boolean retrieval principally requires determination of the presence or absence of terms in

documents, with no regards to important statistics like the frequencies of occurrence of terms within documents or across documents, or the number of documents that contain the various query terms
2. Users find it difficult to construct effective Boolean queries when they use the terms AND, OR, or NOT, which have a different meaning in natural language than the way Boolean search engines use them.
3. Boolean retrieval requires the use of a controlled vocabulary when specifying queries; the terms used in the vocabulary should match terms in the index. Users would prefer to be able to provide free text queries, not constrained by any vocabulary
4. Boolean queries that use the AND operator tend to produce very high precision but unfortunately very low recall results; Boolean queries using the OR operator on the other hand, produce very high recall, but unfortunately low precision results. Queries that result in some middle ground with respect to the documents retrieved would be useful.

With the advent of the Web, and the explosion in the availability of electronic documents, Boolean retrieval has been largely replaced by other retrieval techniques that overcome several of these shortcomings. The Vector-space model (VSM) [1], [5], [6], [7], for example, is much more widely used for IR than Boolean retrieval. In the VSM, each document is represented as an N-dimensional vector of the `N` terms that constitute the index, with each term in a document given a weight that is computed from the statistics of the frequency of occurrence of the terms within the document and across the document collection. The similarity between two documents, where a document could be one of the documents in the collection, a user query, a user profile vector, or any N-dimensional representation of the terms of the document collection, is determined by any one of several similarity measures. Hence, IR using the VSM involves comparing the query vector with each of the documents in the collection, and presenting to the user the documents with highest similarity measures.

The vector space model comprises three stages. The first stage involves document indexing. Because of difficulties involved in working in a high-dimensional space (the so-called 'curse of dimensionality') and because high frequency words in a document do not help much in describing the document content, dimensionality of the document representation is reduced, first by passing the terms of the document collection as parameters of a stop-list algorithm to remove the less descriptive terms, and second, by using a stemming algorithms to ensure that several words with the same stem are treated as the same term.

The second stage assigns weights to the index terms to enhance retrieval of relevant documents when a query is issued. Assignment of term weights assumes that the importance of a term within a document is proportional to the frequency of its occurrence within the document (`TF`) and inversely proportional to its document frequency (`DF`), i.e., the number of documents that contain the term, hence the name 'term frequency × inverse document frequency' (TF × IDF) used to describe VSM. In order to avoid giving preference to longer documents in which terms are likely going to appear more frequently, document lengths are normalized in the model. It is common in IR to represent IDF with log(N/DF), where N is the number of terms in the collection, and used here to scale the IDF score.

The last stage of the model involves determining the similarity between document and query vectors. There are several measures for document-document similarity, including cosine similarity (the most commonly used), Pearson correlation index, Jaccard coefficients, and Dice coefficients. The cosine similarity between two document vectors `d1` and `d2` is given by:

$$Sim(d1, d2) = \frac{\overrightarrow{d1}.\overrightarrow{d2}}{|\overrightarrow{d1}|\ |\overrightarrow{d2}|}$$

The effect of the denominator is to length-normalize the two document vectors so that long documents do not disfavour shorter, but equally relevant documents.

The stages involved in constructing a VSM for a large document collection, and then efficiently searching the model at query time places significant demands on computing resources – both RAM and the processor. Satisfying these demands requires use of data structures and algorithms that efficiently handle the large amounts of data involved.

2. Objectives of Paper

In this work, we consider IR for documents that are represented using the VSM. We pay particular attention to the following:

1. The choice of data structures in the process of creating an index for the documents in a collection, and in the process of representing and using the model
2. The process of weighting individual terms of a document

We use a large data set – the OHSUMED data set [8], to demonstrate the creation of the model.

We note that the VSM considers a document as a bag of words with no importance placed on grammar or word order, and hence does not dwell much on semantics. In spite of this limitation, the model is widely used in IR tasks like search engine design, and scales well to large document collections. This is important because as the data size increases, efficient data structures play a much more significant role than larger RAM in managing complexity.

3. Methodology

3.1 The OHSUMED data set used

The OHSUMED test collection [8] is a set of 348,566 references from MEDLINE, the on-line medical information database, consisting of titles and/or abstracts from 270 medical journals over a five-year period (1987-1991). A small portion of the data set is designed to be used as training data for information filtering tasks; this training set was not indexed in this work.

3.2 Algorithm

Figure 1 is an illustration of the stages involved in generating the VSM for the collection. The objectives are (1) to create an inverted file, which comprises two key components: a dictionary of index terms, and for each term, a list of the documents (i.e., the postings list) the term occurs in; and (2) to create document vectors for each document in the model. The inverted file also has the following term and document statistics that are needed for the computation of term weights within documents:

- Term statistics: For each term, statistics on term frequency across the collection (TF), list of documents containing the term, and for each such document, the term frequency (DTF)
- Document statistics: For each document containing a given term, statistics on the number of terms in the document, the list of terms that constitute the document, and for each such term, the frequency of the term in the document.

Additionally, for each document vector, statistics need to be kept on the following:

- The number of terms in the document

- For each term in the document, the frequency of occurrence of that term in the document

A key difficulty in working with the inverted file is its sheer size: while the terms that constitute the dictionary would normally not overwhelm the RAM of most computers, the postings list is typically too large to fit in RAM. Hence, a balance must be struck between fast main memory processing, and the much slower secondary memory processing.

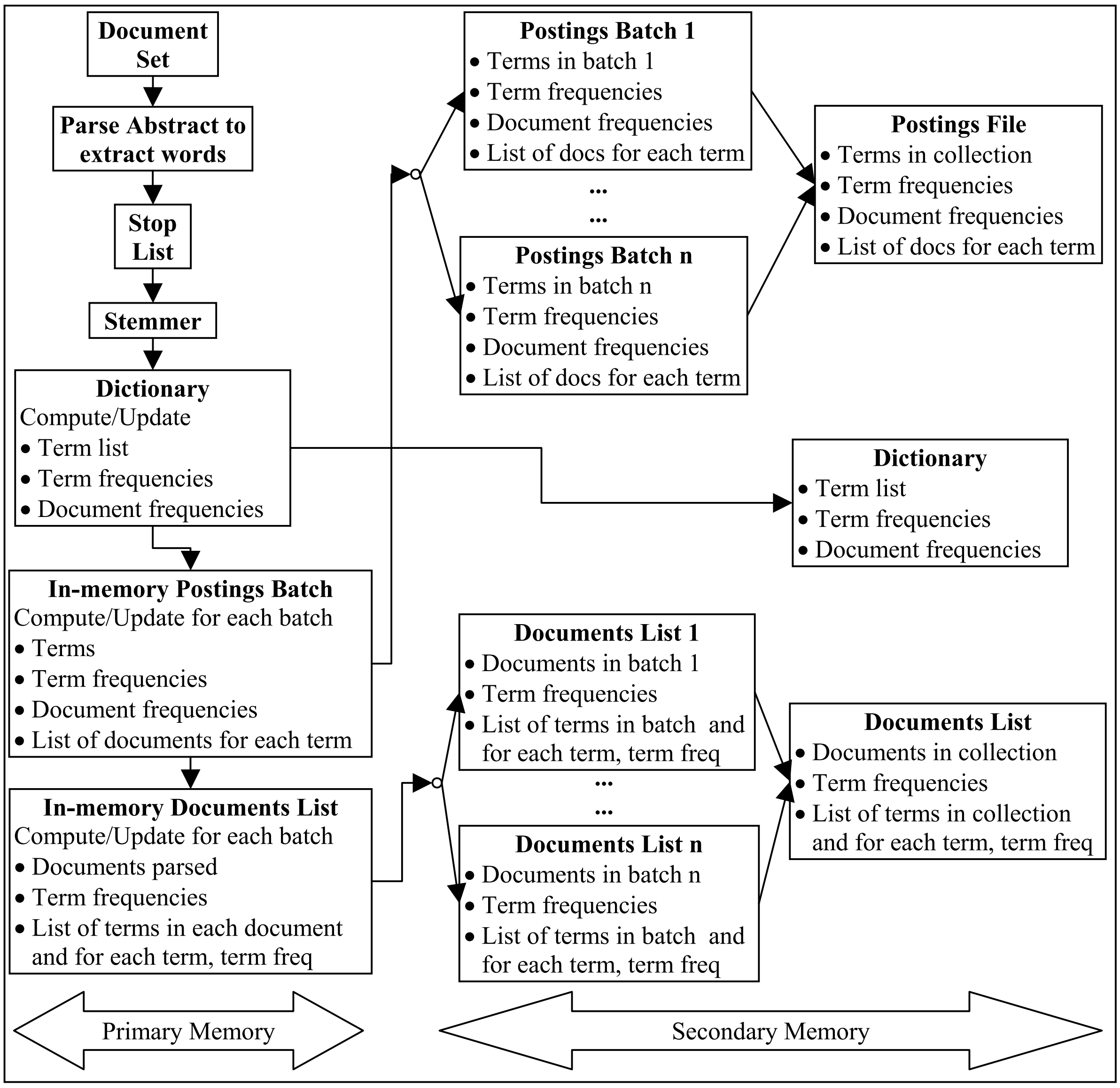


Figure 1: Stages in creating the VSM for the collection

Main memory processing: The activities that are performed in main memory are shown on the left of Figure 1. The process begins by parsing the text of the collection to extract words – interpreted here as any group of characters separated by white space characters or any character in the following set: {0..9 $ % * + - _ . : " , ( ) [ ] = ? & # ! < >} – from the abstracts. Each word found in the Smart stop list [9] is considered a high frequency word, and eliminated. The retained words are then stemmed using a modified version of the original Porter Stemming algorithm [10] to further reduce document dimensionality. The list of stemmed words constitutes the index or dictionary, and for each such word, statistics are collected on term frequency (number of terms in the collection) and document frequency (number of documents containing the term).

The postings list is a listing, together with relevant term statistics, of all the documents that contain each term, and as indicated above, is too big to fit into main memory. Yet, it would be prohibitively slow to process the postings on a term-by-term basis, or even on a document-by-document basis. The same argument holds for the generation of document vectors in the VSM. In this work, our approach was to work with batches of up to 10,000 documents at a time, generate the postings list and document vectors on-line, and then transfer each batch into secondary storage, before starting a new batch.

Secondary memory processing: The price to pay for only creating batches of the postings and document vector files is the need for additional processing of the batches stored in secondary memory. Secondary memory processing involves the following:

- Merging of the postings files into one complete postings file
- Merging of the document vector files into one complete documents file
- Storing of the dictionary in secondary memory after all terms in the collection have been parsed

3.3 Data structures

The data structures used to create the inverted file and document vectors are illustrated in Figure 2.

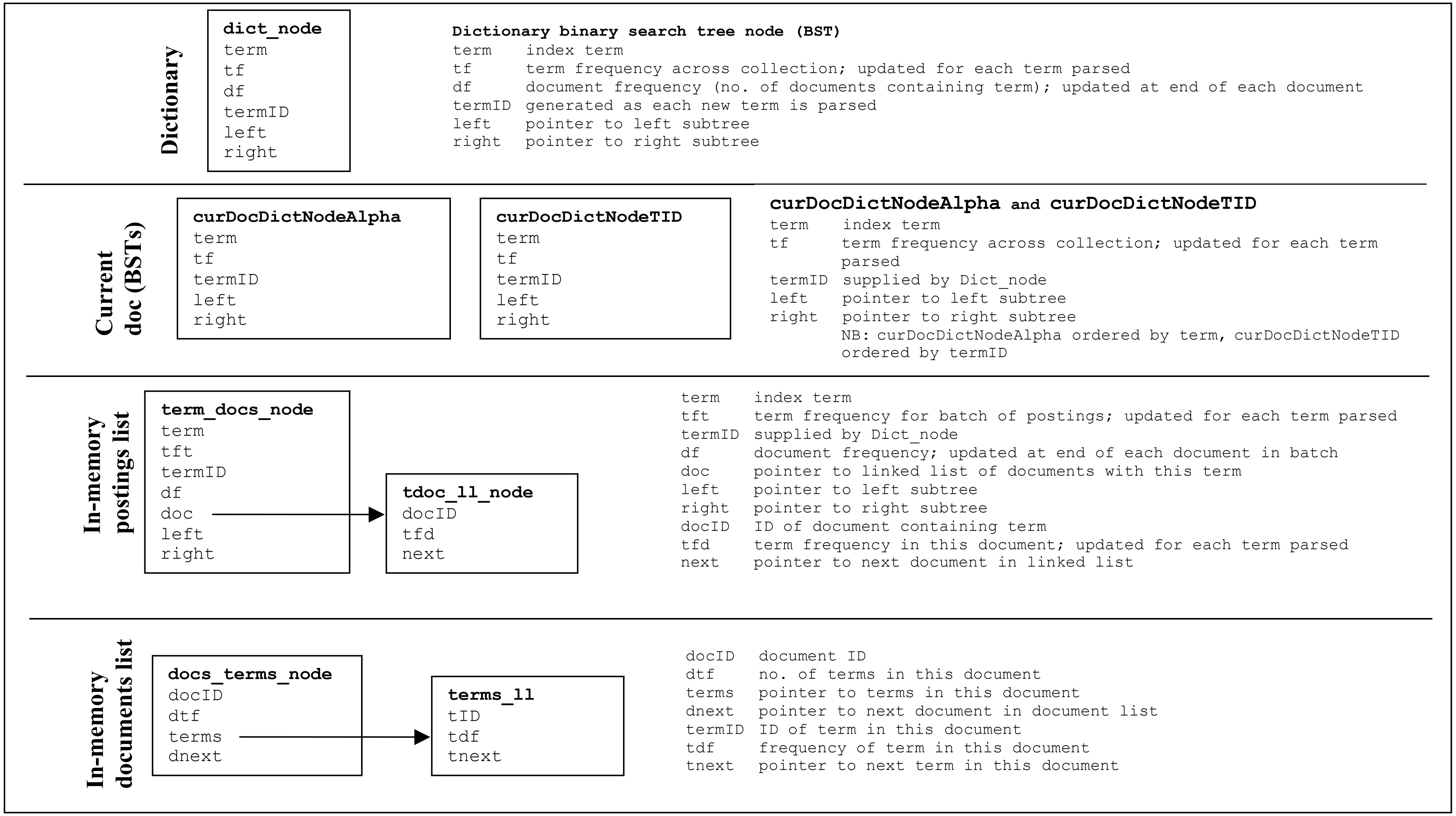
Dictionary: The dictionary is a binary search tree (BST) of terms that is constructed in memory and ordered by index term, as the document collection is parsed. The frequency of each term in a node is updated as each term is parsed; the document frequency for each term on the other hand, is incremented by 1 after the document is parsed.

In-memory postings list: This is a BST of a batch of up to 10,000 document terms. In addition to the statistics that are collected in a dictionary node, each node of this tree has a pointer to a linked list of documents that contain the term. This linked list tracks the documents that contain the term, and for each of these documents, the term frequency in the document. The documents in the list are by default stored in document ID order, and so there is no need for expensive list traversal as new documents are processed.

In-memory documents list: This is a linked list representing a batch of up to 10,000 documents. Each node of the documents list tracks the number of terms in that document and has a pointer to the corresponding document vector, i.e., the terms that are present in the document. For ease of processing, this terms list in stored in term ID order.

Current document BSTs: Two BSTs, `curDocDictNodeAlpha` and `curDocDictNodeTID`, both of them with term nodes are created for each document that is parsed. The only difference between these two BSTs is that the former is ordered by term name and the latter by term ID. As each term in the collection is parsed, the term frequency on these BSTs, and the document list for each term in the in-memory postings list, are updated. Then, at the end of each document parsed:

- curDocDictNodeAlpha is used to update the document frequencies of corresponding terms in both the dictionary BST and the in-memory postings list
- curDocDictNodeTID is used to maintain the terms linked list in the in-memory documents list. The advantage of using curDocDictNodeTID over curDocDictNodeAlpha is that the terms are accessed in term ID order, the same order we would like them in the document vector



Dictionary binary search tree node (BST)

term	index term
tf	term frequency across collection; updated for each term parsed
df	document frequency (no. of documents containing term); updated at end of each document
termID	generated as each new term is parsed
left	pointer to left subtree
right	pointer to right subtree

curDocDictNodeAlpha and curDocDictNodeTID

term	index term
tf	term frequency across collection; updated for each term parsed
termID	supplied by Dict_node
left	pointer to left subtree
right	pointer to right subtree
	NB: curDocDictNodeAlpha ordered by term, curDocDictNodeTID ordered by termID

term	index term
tft	term frequency for batch of postings; updated for each term parsed
termID	supplied by Dict_node
df	document frequency; updated at end of each document in batch
doc	pointer to linked list of documents with this term
left	pointer to left subtree
right	pointer to right subtree
docID	ID of document containing term
tfd	term frequency in this document; updated for each term parsed
next	pointer to next document in linked list

docID	document ID
dtf	no. of terms in this document
terms	pointer to terms in this document
dnext	pointer to next document in document list
termID	ID of term in this document
tdf	frequency of term in this document
tnext	pointer to next term in this document

Figure 2: Data structures used in constructing the VSM

4. Results

Documents: The documents in the training set were not considered in this work. Parsing the remainder of the 348,566 references in the OHSUMED collection revealed that only 190,130 of them contained abstracts; it is these 190,130 documents that were used in this work. Each document is stored using the following notation:

Document = `DOC-ID+SP+TF+:+1{TID+SP+TTF+;}N`

where `DOC-ID` is Document ID, `SP` is space, `TF` is term frequency, `TID` is term ID, `TTF` is frequency of term `TID` in document `DOC-ID`, `N` is the number of distinct terms in the document.

For example, the treatment of document with ID 81944 is illustrated in Figure 3.

Document 81,944 Abstract: The second case occurred of pregnancy in the remaining distal segment of a fallopian tube after segmental resection for an isthmic pregnancy.

	Terms											
Changed by stop list	case	occurred	pregnancy	remaining	distal	segment	fallopian	tube	segmental	resection	isthmic	pregnancy
Changed by stemmer		occur	pregnanc	remain					segment	resect		pregnanc
Distinct terms	case	occur	pregnanc	remain	distal	segment	fallopian	tube		resect	isthmic	
TID & frequency	480 (1)	110 (1)	464 (2)	384 (1)	2,288 (1)	1,453 (2)	5,279 (1)	2,078 (1)		2,073 (1)	12,095 (1)	

Document representation: 81944 12:110 1;384 1;464 2;480 1;1453 2;2073 1;2078 1;2288 1;5279 1;12095 1;

Figure 3: Document representation example

Dictionary: A total of 91,162 index terms were generated. Each index term was stored in secondary memory using the following notation

Index term = `TERM+SP+TID+SP+TF+SP+DF`

where `TERM` is the term, `SP` is space, `TID` is term ID, `TF` is term frequency, and `DF` is document frequency

For example the term 'patient' is stored as: `patient 196 325158 92355`. This term occurs 325,158 times in the document collection and is present in 92,355 documents.

Postings: As mentioned earlier, the postings file was created by merging several batches of postings that were generated in RAM. Altogether, there were 29 batches of postings generated, with an average of 6556 documents per batch. Postings were stored using the following notation:

Posting = `TERM+:+TID+SP+TF+SP+DF+SP+;+1{DID+SP+DTF+;}N`

where `TERM` is the term, `TID` the term ID, `SP` is space, `TF` the term frequency, `DF` the document frequency (number of documents in which the term is found), `DID` the document ID of a document containing the term, `DTF` the frequency of the term in that document, and `N`, the number of distinct documents containing the term.

For example, the term 'zymodem' is represented in the postings list as follows:

zymodem:11133 23 10;57373 2;77378 2;118021 3;177440 2;182587 1;227103 2;266095 5;301635 4;326587 1;330918 1;

This indicates that the term has 11133 as term ID, it occurs 23 times across the document collection and in 10 distinct documents, and the rest of the representation shows pairs of document IDs and term frequencies for the these 10 distinct documents.

5. Importance of Work

An efficient, scalable data handling and processing mechanism is fundamental in digital content and knowledge management systems. This is especially true as documents involved are unstructured and so cannot benefit directly from database technology. Tasks in these systems include clustering, data mining, information filtering, classifier design, information retrieval, and personalization systems design. Researchers in these areas and others can thus benefit from the approach that has been presented in this paper.

6. Conclusions

We have demonstrated in this paper that by making good choices of data structures (principally BSTs and linked lists) and by making reasonable compromises in the face of limited main memory by relegating some processing to secondary storage, the dictionary and document files can efficiently be generated for reasonably large document collections like the OHSUMED data set. With ever increasing data set sizes, computers endowed with even greater memory sizes will help the process to some extent. But perhaps a more productive approach would be to consider other data structures like B-trees, which should be able to read and write much larger blocks of data than binary search trees.

References

[1] C. D. Manning, P. Raghavan, and H. Schutze, *An introduction to information retrieval*, Online. Cambridge, England: Cambridge University Press, 2009.
[2] N. Belkin and B. Croft, "Information filtering and information retrieval: two sides of the same coin?," *Communications of the ACM*, vol. 35, no. 12, pp. 29–38, Dec-1992.
[3] W. Cooper, "Getting beyond Boole," *Inf. Process. Manag.*, vol. 24, no. 3, pp. 243–248, 1988.
[4] E. Fox and M. Koll, "Practical enhanced Boolean retrieval: experiences with SMART and SIRE systems," *Inf. Process. Manag.*, vol. 24, no. 3, pp. 257–267, 1988.
[5] G. Salton, *Automatic text processing*. Reading, Massachusetts: Addison-Wesley, 1989.
[6] G. Salton and W. J. McGill, *Introduction to modern information retrieval*. New York: McGraw-Hill, 1983.
[7] G. Salton and A. Wong, "A vector space model for automatic indexing," *Commun. Acm*, no. 18, 1975.
[8] trec@nist.gov, "TREC-9 filtering track collections." Jul-2007.
[9] "Smart stop list." .
[10] M. F. Porter, "An Algorithm for suffix stripping," *Program*, vol. 14, no. 3, pp. 130–137, 1980.