Compressing Inverted Files in Scalable Information Systems by Binary Decision Diagram Encoding[*†]

Chung-Hung Lai and Tien-Fu Chen

Department of Computer Science
National Chung Cheng University
Chiayi, Taiwan 621, ROC

chen@cs.ccu.edu.tw

Abstract

One of the key challenges of managing very huge volumes of data in scalable Information retrieval systems is providing fast access through keyword searches. The major data structure in the information retrieval system is an inverted file, which records the positions of each term in the documents. When the information set substantially grows, the number of terms and documents are significantly increased as well as the size of the inverted files.

Approaches to reduce the inverted file without sacrificing the query efficiency are important to the success of scalable information systems. In this paper, we propose a compression approach by using Binary Decision Diagram Encoding (BDD) so that all possible ordering correlation among large amount of documents will be extracted to minimize the posting representation. Another advantage of using BDD is that BDD expressions can efficiently perform Boolean queries, which are very common in retrieval systems. Experiment results show that the compression ratios of the inverted files have been improved significantly by the BDD scheme.

Keywords: Inverted File, Scalable Information Systems, BDD, Information Retrieval

[*]This work was supported by National Science Council of R.O.C. under grant NSC 89-2213-E-194-007.

[†]Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage, and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. SC2001 November 2001, Denver (c) 2001 ACM 1-58113-293-X/01/0011 $5.00

1 Introduction

The applications of the widely used information retrieval systems include electronic libraries, on-line news servers, patent querying systems, and legal precedent querying system. The basic structure of the information retrieval system is based on indices for every possible querying target terms. The main problem of the information retrieval system is that when the number of documents in the collection increases, the size of data indices grows significantly. This fact results in longer querying time and more space required to store the indices. Several schemes have been proposed to deal with this problem. These schemes include transforming the document number of each index into the difference of the adjacent documents to shrink the numbers such as d-gap[10], and compressing the indices by encoding those differences of document numbers.

As these schemes can reduce the size of the indices, the time required to perform queries on those compressed indices will be even larger, especially in computing complex Boolean query expressions. In this paper, we propose a compression scheme by using Binary Decision Diagram Encoding (BDD). The BDD scheme can contribute a better compress ratio and improve the querying time for the information retrieval system. The basic idea of the BDD scheme is to transform the traditional inverted lists into a binary function representation. As the document ID's in an inverted list are expected to be random in some sense, compressing schemes of using document locality (e.g., d-gap[10]) may not have benefits for some collections. However, the BDD may extract all possible ordering correlation among large amount of documents and the posting representation can be minimized via logic optimization. Another advantage of using BDD is that BDD expressions can perform Boolean queries efficiently, which are very common in retrieval systems. Experiment results show that the compression ratios of the inverted files can be improved significantly by the BDD scheme.

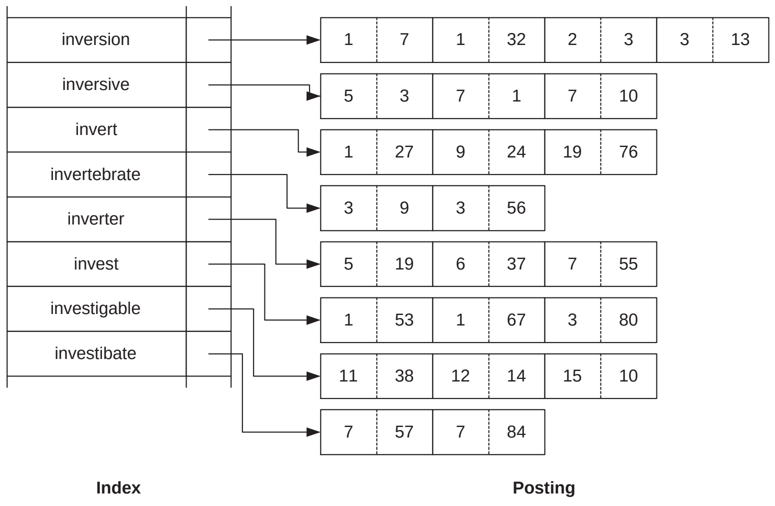
The remainder of the paper is organized as follows. In Section 2, we will give the necessary background for inverted file and BDD. Section 3 shows the compression scheme by BDD and its variation. In section 4, performance results are given. We show the benchmarks and compression ratios. Section 5 concludes this study.

2 Background

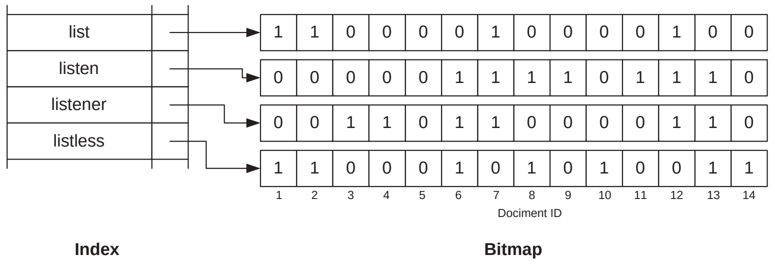
With the popularization of computers and the Internet, the measure of electronic information is growing larger and larger. Witten et. al. [14] estimate that there are more than 6,000 gigabytes of information circulating on the Internet, and the amount of information is doubled every half-year. In order to search electronic information efficiently and reduce the time and workload for querying operations, scalable information retrieval systems are mandatory for users to efficiently retrieve required information. Faloutsos et. al. [6] point out that besides performing full-text search to the document collections, information retrieval systems generally use signature file and inverted file to assist the querying operations.

The inverted file basically follow the concept of the indices used in books. Information retrieval systems usually construct indices for the contents of the documents to simplify the querying operations. When the users perform queries, user queries can be satisfied by returning the document pointers whose documents contain requested terms[4, 14].

Figure 1 shows the data structure of an inverted file. The terms of the index are sorted in alphabetical order. To increase the searching speed, we can use B-tree [5], TRIEs, hashing table, or other data structures to reorder the terms in the index. For each term we maintain a data structure called posting, which is a list of pointers pointing to documents containing the term, and the positions of the term appeared in those documents. Alternatively, we can also use bitmap as the data structure of the posting as shown in Figure 1(b). Each term in the index correspond to a bitmap vector. The length of each bit vector equals to the total numbers of document in the collection, and each document corresponded to a position in the bit vector.



(a) Data structure of posting with document list



(b) Data structure of posting with bitmap

Figure 1: Inverted Files

Rmit et. al. [13] compare the signature file and the inverted file. They discover that in terms of querying time, used space, and provided functions, the inverted files performs better than the signature files. Riloff et. al. [12] also state that most information retrieval systems choose inverted files as basic data structures. Because representing the inverted list by bitmaps often require a lot space due to the sparsity of each bit vector, Moffat et. al. [9] proposed the parameterized compression method, which use different encoding to encode each bit vector according its characteristics.

When the queries are made to search for more than one terms, the searching operations often involve Boolean operations, which combine the requested terms with Boolean operators, such as AND, OR, and NOT. After parsing the requested sentence for the relation of the terms, the system locates each term, perform Boolean operations on the inverted list, and return the result pointer list for the response of the user's query. Moffat et al [10] introduce the self-indexing inverted file to speed up the Boolean queries and ranked queries in the inverted files.

In the area of logic synthesis, binary decision diagram (BDD) is a simple yet powerful data structure for Boolean operations. The advantages of BDD have been examined by Brayton [1] and BDD is widely applied in the synthesis and verification research areas[7]. A BDD is a directed acyclic graph representing a switching function. There are two types of nodes in the BDD: *decision nodes* and *terminal nodes*. *Decision nodes* are the nodes that have outgoing edges connecting to other decision nodes or terminal nodes. Each decision node is marked with a variable, and the direction of the outgoing edge of a decision node is determined by the value of that variable. *Terminal nodes* have no outgoing edges, and contain fixed value represent the result of the function. Because BDD is always used at Boolean operations, the outgoing edges of decision nodes can be either 1-edge or 0-edge, and the values of the terminal nodes can be either 1 or 0.

The BDD of a function is generated by applying the Shannon Decomposition, where the function is generated from inverted lists[2]. With the Shannon Decomposition, the function f can be represented as:

$$f(x_1, x_2, ..., x_n) = \bar{x_1} f(0, x_2, ..., x_n) + x_1 f(1, x_2, ..., x_n) \tag{1}$$

By recursively applying the Shannon Decomposition to the function, we can construct the BDD of the function, where each node is associated with a variable, and each edge of that node are connected to one of the two decomposed functions.

To reduce the size of BDD and make the BDD canonical for easy operation, the BDD is transformed into reduced ordered BDD (ROBDD) by removing the redundant nodes and merging isomorphic sub-trees. Figure 2 (a) shows an example of BDD with the function $f = x_1x_3 + x_2x_3$, and Figure 2 (b) shows the ROBDD corresponding to the BDD in Figure 2 (a).

The Boolean operations of the BDD's are evaluated by the if-then-else operator (ITE). ITE is a ternary operator, and all two-augment operators, like AND, OR, NO, $\langle$, $\rangle$, can be expressed in terms of ITE [7]. The value of a function is recursively applied the ITE algorithm

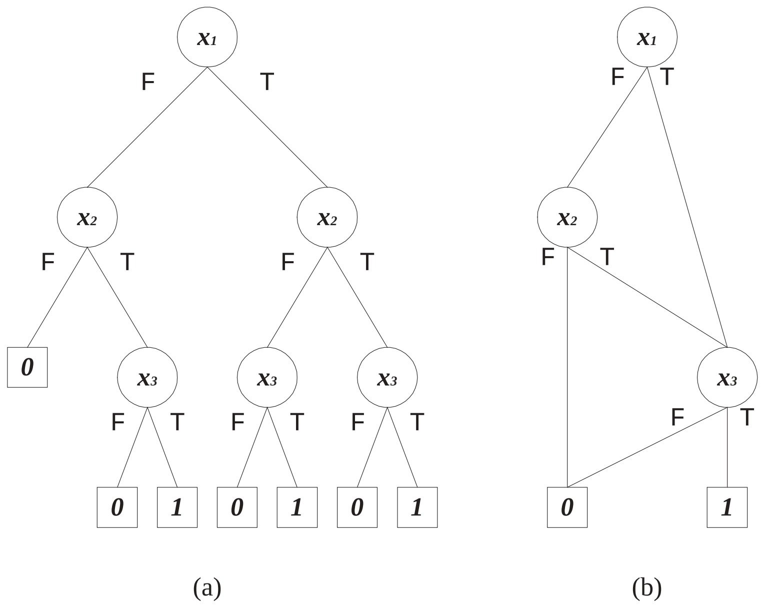


Figure 2: Example of $f = x_1x_3 + x_2x_3$ by BDD

until the terminal nodes are met. After the recursive formulation of ITE is completed, the resulted BDD can be constructed.

3 Compressing the Inverted File by BDD

3.1 Transforming list into BDD

To transform an inverted list into a BDD representation, first, we have to find the logic function representing this inverted list. We consider the presence of a document in the inverted list as true, while the absence of a document in the inverted list as false. Thus, by listing all possible documents IDs encoded in binary code as input, and set the output as the presence of each document in the inverted list, we can generate a truth table for the inverted list. For example, the inverted list ⟨7: 0, 3, 5, 8, 9, 12, 14⟩ can be transformed to the truth table as shown in Figure 3 (a).

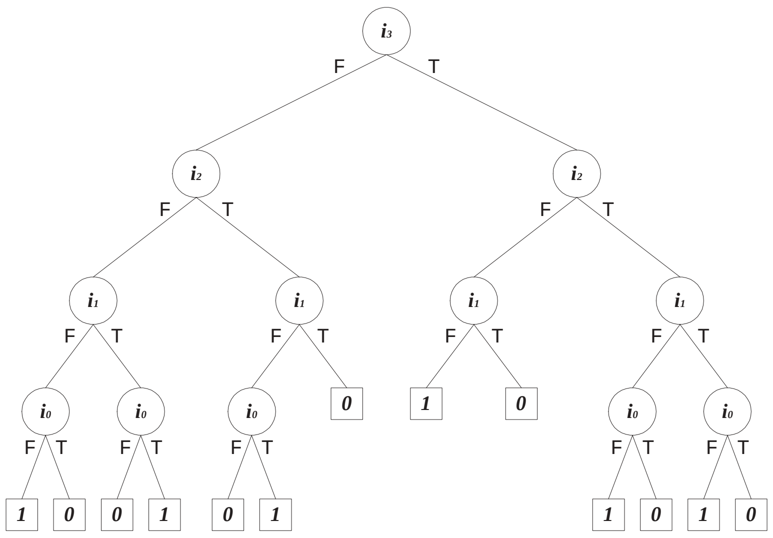
From the truth table, we can generate the Boolean function of the inverted list. For example, the truth table in Figure 3 can be represented as the Boolean function is the sum of product term as:

$$f = i_2\bar{i}_1\bar{i}_0 + \bar{i}_2 i_1 i_0 + i_3\bar{i}_2\bar{i}_1 + i_3 i_2\bar{i}_0 + \bar{i}_3\bar{i}_2 i_1 i_0 + \bar{i}_3 i_2\bar{i}_1 i_0 \qquad (2)$$

This function can be represented as the non-reduced BDD shown in Figure 3.

i_3	i_2	i_1	i_0	f
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0

(a) A truth table from an inverted list



(b) BDD generated

Figure 3: Example of generating BDD from inverted list

Since most operations in the information retrieval systems with inverted files involve Boolean expressions, we can use BDD to perform Boolean operations to speedup the queries. As now we want to store the transformed BDDs as the data structure of the indices, the key issue becomes how to encode and decode BDD efficiently using minimized space.

3.2 Encoding and decoding BDD

Because the structure of the non-reduced BDD is more regular and the Reduced Ordered BDD requires more pointers to store, we represent the inverted list in the form of non-reduced BDD to reduce the size of the inverted file. From Figure 3 we can see that the non-reduced BDD is, in fact, a binary tree, consisting of *decision nodes* with two branches being true and false and *terminal nodes* giving the output of either 0 or 1.

Column	Bit string
Decision Nodes	111111111000110000000000
Terminals nodes	1001010101010
Total bit length	37

Figure 4: Example of BDD encoding string

We can represent a BDD by using two segments. The first segment indicates the type of the child nodes, where a child decision node is represented by 1, and a child terminal node is represented by 0. The second segment represents the output value of each terminal node, which is either 1 or 0. The encoding can be performed via a breadth-first search to traverse the non-reduced BDD and outputting the one-bit encoding of the branches. Then we output the value of each terminal node from the left most terminal node to the right. With this encoding scheme, the non-reduced BDD given in Figure 3 can be represented as shown in Figure 4.

Since the order of all elements in the BDDs of an inverted list is fixed, the input nodes represented at each level are known. Reconstructing the BDD from the encoding string is also a simple task. From the root node, we parse the first two bits of the child type bit string to decide whether the right and left child is a decision node or a terminal node. Then for the nodes at the second level, we scan the bit string again to decide the type of each child node. The binary tree is constructed when each bit in the child type bit string is determined. After constructing the binary tree, we fill the value of each terminal node according to the terminal node bit string. Figure 5 illustrates the reconstruction of each level of the BDD given in Figure 3 based on the bit strings in Figure 4.

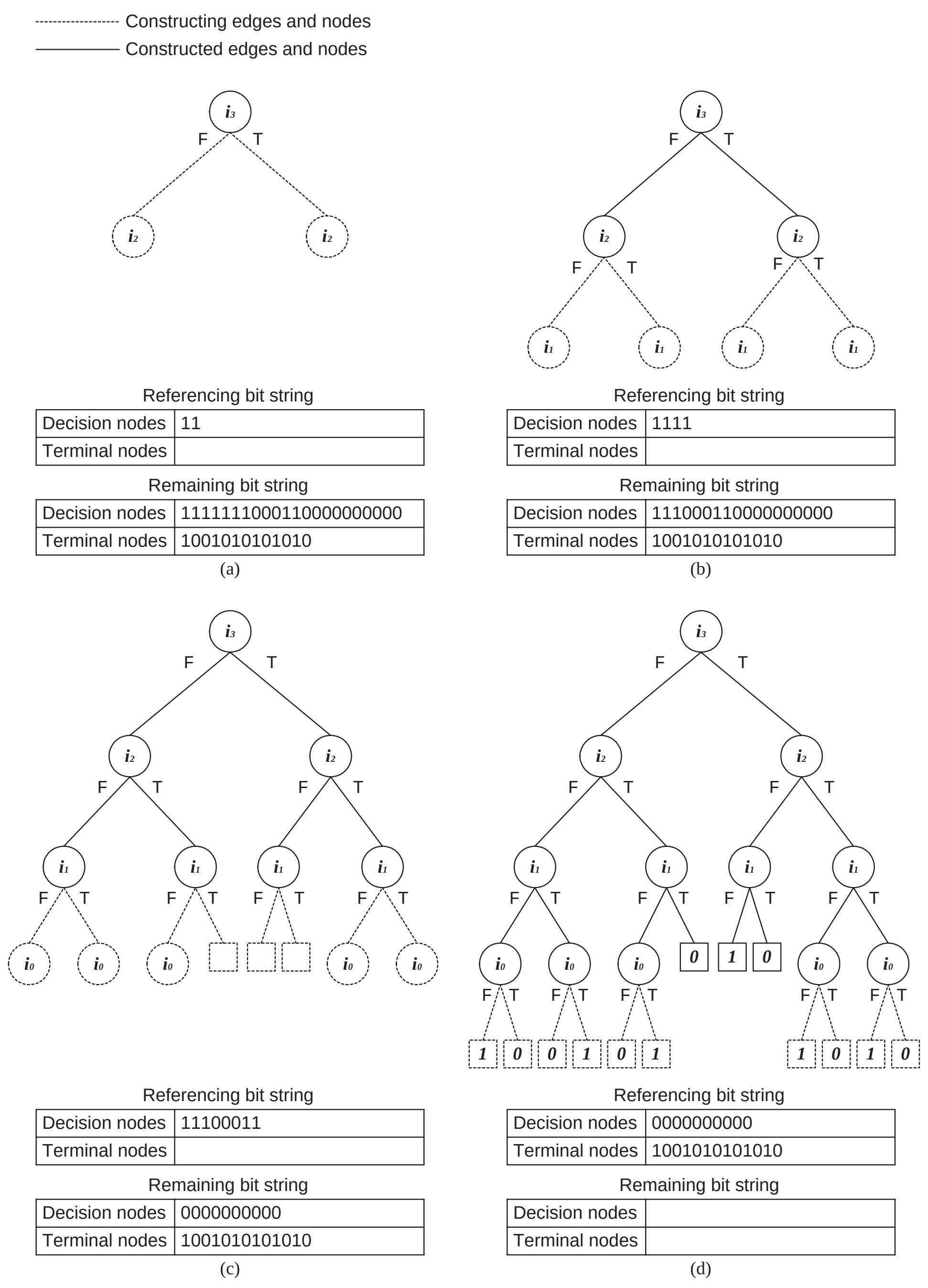
3.3 Operations of the BDD inverted list

To determine whether a certain document exists in an inverted list, we can simply traverse the BDD representation of the inverted list according the value of the document ID. To perform queries that are looking for a specific term, the system can enumerate the document lists by traversing the BDD with deep-first search. If the queries consist Boolean operations, we may transform the non-reduced BDD for each single term into reduced ordered BDD, and then apply those BDD's can be merged by the ITE function with Boolean operations. Then the resulted ROBDD is traversed to obtain the final document list as the query results [3, 7].

3.4 Use partitioned BDD to reduce space

When the document collection is very huge, the BDDs used to encode the inverted lists may consist of many levels. In case that there are only a few documents in a specific inverted list and the logic function of the inverted list is simple, representing a few document ID's in BDD will be space-inefficient. Narayan et al [11] suggest that the partitioned BDD is a better method to reduce the size of BDD.

However, because the locality characteristic of the inverted list, the partitioned BDD method is also suitable for reducing the size of the inverted list transformed BDD. For example, the bit string used to encode the BDD in Figure 3 is 37 bits long. However, if we partition the inverted list into four partitions as illustrated in Figure 6 (a), the partitioned BDD is shown in Figure 6 (b), and the bit strings to encode the partitioned BDD is shown in Figure 6 (c), which is 31 bits long.



Referencing bit string

Decision nodes	11
Terminal nodes	

Remaining bit string

Decision nodes	1111111000110000000000
Terminal nodes	1001010101010

(a)

Referencing bit string

Decision nodes	1111
Terminal nodes	

Remaining bit string

Decision nodes	111000110000000000
Terminal nodes	1001010101010

(b)

Referencing bit string

Decision nodes	11100011
Terminal nodes	

Remaining bit string

Decision nodes	0000000000
Terminal nodes	1001010101010

(c)

Referencing bit string

Decision nodes	0000000000
Terminal nodes	1001010101010

Remaining bit string

Decision nodes	
Terminal nodes	

(d)

Figure 5: Example of reconstructing BDD

4 Performance Evaluation

To show the compress efficiency of our inverted file compress schemes, we perform experiments with three text collections. We use the MG system provided by Witten et al.[14] to generate the inverted list.

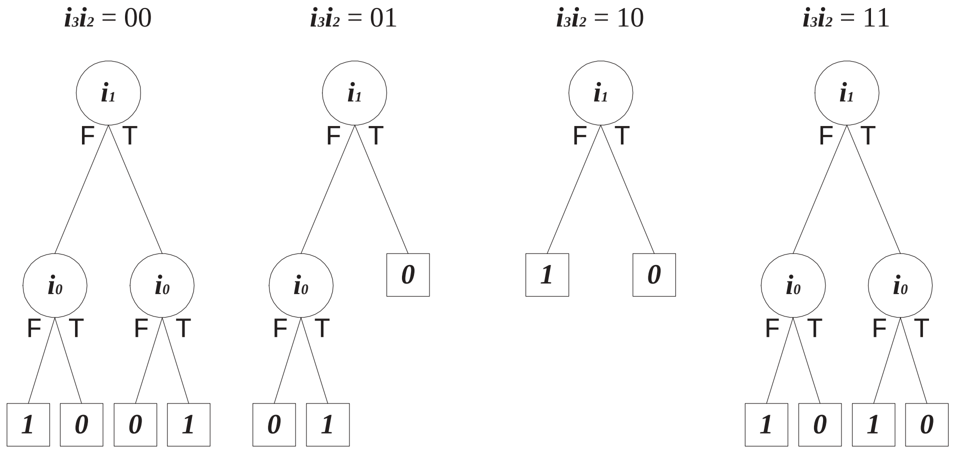
$i_3i_2 = 00$		
i_1	i_2	f_{00}
0	0	1
0	1	0
1	0	0
1	1	1

$i_3i_2 = 01$		
i_1	i_2	f_{01}
0	0	0
0	1	1
1	0	0
1	1	0

$i_3i_2 = 10$		
i_1	i_2	f_{10}
0	0	1
0	1	1
1	0	0
1	1	0

$i_3i_2 = 11$		
i_1	i_2	f_{11}
0	0	1
0	1	0
1	0	1
1	1	0

(a) Example of a truth table from partitioned inverted list



(b) Partitioned BDD generated from truth table

	Bit strings			
Column	$i_3i_2 = 00$	$i_3i_2 = 01$	$i_3i_2 = 10$	$i_3i_2 = 11$
Child type	110000	1000	00	110000
Terminal node	1001	010	10	1010
Total bit length	31			

(c) Encoding results

Figure 6: Encoding partitioned BDD

We choose three benchmarks: "PAPER", "FBIS", and "LATIMES" as our experiment samples. The collection "PAPER" is a large collection of papers from various proceedings and journals. The "PAPER" collection represents a benchmark containing smaller amount of documents with a larger size of content in each document. Both "FBIS" and "LATIMES" collections are contained in the TREC (TExt Retrieval Conference) Information-Retrieval Text Research Collection[8] provides by NIST (National Institute of Standards and Technology) of U.S. The TREC collections are widely used for researching and evaluating information retrieval systems. The "FBIS" collection consists documents from the Foreign Broadcast Information Service, and the "LATIMES" collection consists documents from Los Angeles Times. Both "FBIS" and "LATIMES" come in SGML format formulated by NIST. The statistics of these text collections we used are listed in Figure 7.

For the BDD operations, we use the library in the BDD package of US Berkeley Synthesis Package (SIS). We estimate the size of a BDD generated from an inverted list by calculating the number of branches of a BDD. For each inverted list in the inverted file, we transform it

	Collections		
	PAPER	FBIS	LATIMES
Documents	6,502	130,963	132,626
Total terms	34,905,485	72,922,896	72,101,368
Distinct terms	560,083	337,331	343,582
Total Size (MB)	183.15	442.52	412.92

Figure 7: The text collections

into a programmable logic array (PLA) in the format of blif file, which is used by SIS. Then we use functions in the BDD package to generate the non-reduced BDD, and calculate the number of branches and leaves to estimate the size required to store the BDD. Since we use one bit to represent the type of a branch and one bit to indicate the value of the leaf, the size of the BDD is calculated by adding the number of branches to the number of leaves.

To estimate the size of the partitioned BDD, we transform inverted lists into several programmable logic arrays according to the number of partitions. We divide the document ID domain equally according to the value of the Document ID. For example, if we divide an inverted list with at most 16 documents to four partitions, then each partition contains 4 documents, and the partitions are document 1 to 4, 5 to 8, 9 to 12, and 13 to 16. After we generate the programmable logic array for each equally divided partition, we generate BDD for each programmable logic array and calculate the size these BDDs to estimate the size of the inverted list stored in partitioned BDD.

4.1 Experiment Results

We divide each of the document collections into 2, 4, 8 and 16 partitions, and generate non-reduced BDD for each partition to estimate the size of the partitioned BDD inverted file. The divisions of the document collections are according to the most significant bits of the binary encoded document ID. That is, if we divide the document collection into eight partitions, the top most three bits of the binary encoded document ID are used to decide which partition the document is located.

We use the binary encoded inverted file as the basis for a comparison with each collection. Figure 8 shows the compression ratio of the BDD inverted file and the 2, 4, 8, and 16 partitioned BDD inverted file. Table 1 lists the average size in bits and the compression ratio of an inverted list, or posting, of the inverted files encoded in each BDD scheme for each of the three document collections.

Because the inverted lists of large document collections are generally larger than that of small document collections, the inverted lists in large document collections provides more elements for logic simplification to simplify the logic functions generated from the document IDs in the inverted lists. Thus although the BDD representation of inverted lists in small document collection have less levels, the simpler functions of inverted lists in large document

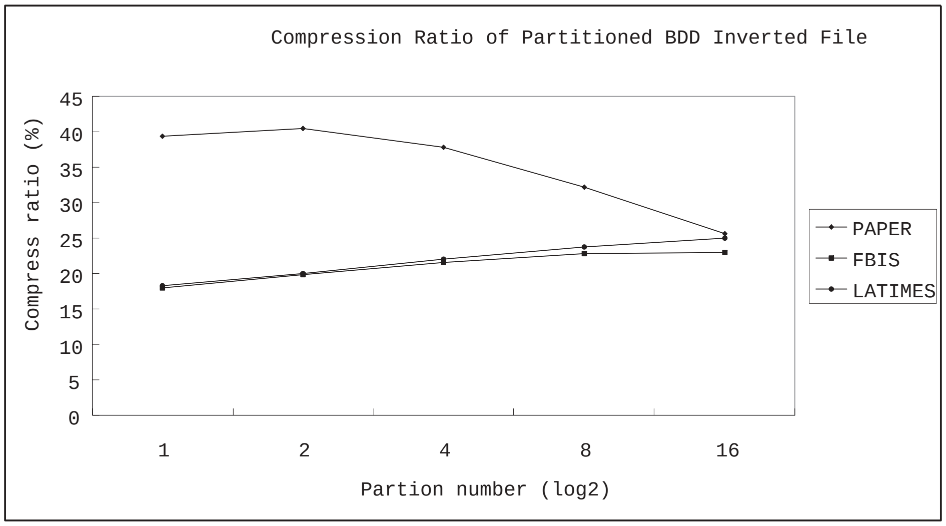


Figure 8: Compression ratios of three document collections encoded in partitioned BDD

collections lead to simpler BDD representations. This is shown in the compression ratio comparison in Table 1, where the "PAPER" collection is a small document collection with 6,502 documents, and the "FBIS" and "LATIMES" collection are both large document collections with more than 130,000 documents.

The partitioned BDD performs well when the logic functions of one or more partitions are very simple. For partitioned BDD with small partition numbers, because the documents in the inverted list are dispersed among several partitions, the size of the partitioned BDD encoding inverted list is no better than the original BDD encoded inverted list. The situation is worse when the logic function of each partitioned BDD is complex. When the number of partitioned BDD is getting larger, there will be partitions containing no or few documents, which makes the logic functions of these partitions being simple. In this case, the compression ratio of partitioned BDD encoded inverted list is better than the original BDD encoded inverted list. Figure 8 shows that with the number of partitions increasing, the compression ratio of the "PAPER" collection is first increased then decreased. Because "FBIS" and "LATIMES" are large document collection, although the compression ratio of "FBIS" is reaching the worse point at 16 partitions, both collection require more partitions to reduce the size of the BDD encoded inverted list.

5 Conclusions

In this paper, we propose a new compression scheme to reduce the huge size of the inverted files in a large information retrieval system without loosing the querying efficiency. The basic idea is to transform the inverted list into a logic function, to represent that function in the form of BDD, and then store the BDD directly in the inverted file. It is well known

	PAPER		**FBIS**		**LATIMES**	
Encoding scheme	**posting size**	**comp. ratio**	**posting size**	**comp. ratio**	**posting size**	**comp. ratio**
Binary	213.88	100.00%	2289.56	100.00%	3494.99	100.00%
BDD	84.30	39.42%	410.13	17.91%	638.29	18.26%
2 partitioned BDDs	86.66	40.52%	455.50	19.89%	701.33	20.07%
4 partitioned BDDs	80.98	37.86%	492.70	21.52%	767.30	21.95%
8 partitioned BDDs	68.69	32.12%	521.52	22.78%	830.58	23.76%
16 partitioned BDDs	54.89	25.66%	524.11	22.89%	876.27	25.07%

Table 1: Average posting size of BDD inverted file

that performing Boolean operations with BDDs is very efficient. With the data structure we designed, the size of the BDD inverted list is also competitive. We apply the partitioned BDD scheme to further reduce the size of the BDD inverted list. Experiment result shows that the compression ratios of the inverted files encode in BDD and partitioned BDD inverted file are very good, especially for large data collections.

References

[1] R. K. Brayton, G. D. Hachtel, C. T. McMullen, and A. Sangiovanni-Vincentelli. *Logic Minimization Algorithms for VLSI Synthesis.* Kluwer Academic Publishers, 1984.

[2] F. M. Brown. *Boolean Reasoning: The Logic of Boolean Equations.* Kluwer Academic Publishers, 1990.

[3] R. E. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computer Surveys*, 24(3):293–318, September 1992.

[4] Edited by W. B. Franks and R. Baeza-Yates. *Information Retrieval - Data Structures and Algorithms.* Prantice Hall, 1992.

[5] C. Faloutsos and H. V. Jagdish. On B-tree indices for skewed distributions. In *Prod. 18th Vary Large Database Conf. Vancouver, British Columbia*, pages 363–374, August 1992.

[6] C. Faloutsos and D. W. Oard. A survey of information retrieval and filtering methods. Technical Report CS-TR-3514, Department of Computer Science, University of Maryland, August 1995.

[7] G. D. Hachtel and F. Somenzi. *Logic Synthesis and Verification Algorithms.* Kluwer Academic Publishers, 1996.

[8] D. K. Harman. Overview of the second text retrieval conference (TREC-2). *Information Processing and Management*, 31(3):271–289, May 1995.

[9] A. Moffat and J. Zobel. Parameterised compression for sparse bitmaps. In *Proc. 5th Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 274–285, 1992.

[10] A. Moffat and J. Z. Zobel. Self-indexing inverted files for fast text retrieval. *ACM Trac. Information System*, 14(4):349–379, October 1996.

[11] A. Narayan. Recent advances in BDD based representations for boolean functions: A survey. In *Proc. 12th Intl. Conf. on VLSI Design*, pages 408–413, January 1999.

[12] E. Riloff and L. Hollaar. Text database and information retrieval. *ACM Computer Surveys*, 28(1):133–135, March 1996.

[13] J. Z. Rmit, A. Moffat, and K. Ramamohanarao. Inverted files versus signature files for text indexing. *ACM Trac. Database System*, 23(4):453–490, December 1998.

[14] I. H. Witten, A. Moffat, and T. C. Bell. *Managing Gigabytes - Compressing and Indexing Documents and Images, 2nd Ed.* Morgan Kaufmann Publishers, Inc., 1999.