S-Index Implementation and Performance for Textbases

Ilias Nitsos[1], Dimitris Dervos[2], Georgios Evangelidis[1]

[1] Department of Applied Informatics, University of Macedonia, Thessaloniki, Pcode, GREECE
[2] Department of Information Technology, TEI, Thessaloniki, Pcode, GREECE

Abstract. In the present study we implement an improved, compressed variation of S-Index and consider its performance against a 130 MB textbase. S-Index, where "S" stands for signature and "Index" implies the inverted file index, is a hybrid indexing scheme found to combine advantages from two popular indexing methods: the inverted file and the signature file/bitmap. We present the file structure and consider implementation details relating to the improved compressed variation of S-Index. The results obtained are compared to those of the self-indexed compressed variation of the inverted file index. With minimal configuration/tuning effort, S-Index is measured to occupy less than 5% of the indexed textbase corpus. The latter implies performance comparable to that of file inversion.

1 Introduction

In today's Internet based world of information exchange and dissemination, data are registered in structures that support the underlying model of information representation and processing. Information processing systems range from the traditional database management system (DBMS) [7] [7], to textbase and hypertext Information Retrieval (IR) systems [3, 17] [17, 3], as well as systems which process complex and multimedia data [1, 15] [1, 15]. The modern object relational database system includes extenders that manage complex data and offer advanced query-processing services [12] [12]. Common to all, is the need to apply and utilize appropriate information indexing schemes at the physical level.

File inversion [2, 11] [2, 11] and signature files [8, 16] [8, 16] comprise the two most popular indexing methodologies today. Each one method has its pros and cons, with the inverted index excelling in query processing efficiency (single terms queries, in particular), and the signature files being most efficient in the evaluation of set predicates, often required in handling complex objects in Object Oriented database systems [13] [13]. In the IR bibliography, a number of research papers report on variations that improve the performance of either method. For text indexing in particular, a self-indexing inverted index variation [14] [14] is reported to outperform the superimposed coding based signature file (SC-SF), both in storage utilization as well as in query processing efficiency [18] [18].

The self-indexing inverted index variation in [14] [14] is based on the fact that the inverted list identifiers are stored sorted and can therefore be represented using variable-length codes. This allows for the inverted lists to be compressed. Compressed inverted files are reported to utilize around 5-10% of the size of the textbase being indexed. The reduction in the required disk space for the index is significant, since the corresponding figure for plain inverted files is 30-60% [18] [18].

The S-Index method comprises a hybrid-indexing scheme reported to combine advantages from both worlds, namely: file inversion and signature files [6] [6]. More specifically, it integrates gracefully the indexing of some textbase sections via file inversions and of some others via bitmaps. As it is explained in the sequel, the latter constitute a special case of SC-SF type block signatures. In this respect, S-Index comprises a unifying index structure and this is the reason why it has been given the specific name: 'S' refers to signature, whereas 'Index' implies the inverted index.

The Bitmap or Exactly Reversible Signature File (ERSF) constitutes a particular instance of SC-SF having m = 1, F = V plus the restriction that distinct words correspond to distinct signatures [4] [4]. Using a minimal perfect hash function (MPHF) [9, 10] [9, 10], or a B+ tree, a unique number in [0,...,V-1] can be assigned to each distinct word of the vocabulary. Then, the signature of the word that corresponds to number k (k in [0,...,V-1]) is a unique V-bit signature having only its k_{th} bit set to 1 and the remaining V-1 bits set to 0. The signature for an entire block is created by OR-ing the unique signatures of the D distinct words of the block. In the end, the block signature has D bits set to 1 and V-D bits set to 0. An evident and less complex way for constructing the signature for a block is to set the D bits, that correspond to the D distinct numbers assigned to each of the D distinct words in the block, to 1 and the remaining V-D bits to 0 (see Table 1 for an example). For a single word query the number k corresponding to the word is retrieved with the help of a B+ tree or an MPHF. If a block signature has its k_{th} bit set to one then the block is retrieved, otherwise it is rejected. By construction, all blocks retrieved qualify for the given query, i.e., there are no false drops.

Word	**Word Number**
free	3
text	9
database	12
example	0
Block Signature	1001000001001000

Table 1. A Bitmap example having V = 16 and D = 4.

The main principle behind S-Index (the structure and the functionality of which are to be presented in Section 2 which follows) is that ERSF is found to achieve maximum information compression when D (also called the blocking factor) is equal to or greater than half the size of the vocabulary V. Figure 1 illustrates the ERSF storage utilization curve next to the one of Perfect Encoding (PE) for various D values in the case of a sample textbase of N=1000 words that utilizes a vocabulary of V=400 distinct words. Perfect encoding was introduced in [5] [5], where it is found to achieve the maximum possible information compression for bitmaps.

In short, for each D (distinct) word logical block of text an ERSF signature is generated. This signature is not registered as such, but it is partitioned into fixed length partitions each one of which is taken to propagate down the levels of a (backbone) binary tree structure. Signature sections are appended to various nodes of this structure in such a way so that the number of 1s in each such section equals to or exceeds the number of 0s. In this respect, the scheme stores signature sections in accordance with the principle illustrated in Figure 1.

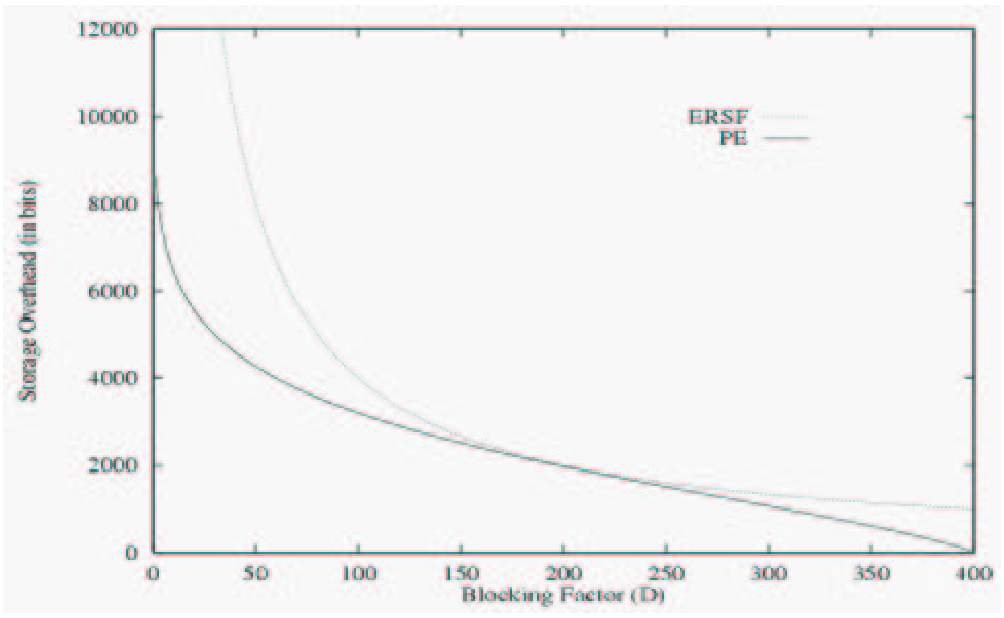

Fig. 1. ERSF and PE storage overhead for different values of D when (V,N)=(400,1000).

Up until today, S-Index has been studied by means of analytical calculations (with certain assumptions on the distributions of "terms" in the "textbase"), as well as by utilizing a relational database management system to simulate its performance on a synthetic textbase as well as on real text [4, 6] [4, 6]. In the sections that follow we introduce a variation of S-Index which compresses the size of the index by reducing the size of the tree structure by one level, plus by avoiding to store signature patterns at the new lowest leaf level. The data structure and the functionality of new variation are considered in detail as well as by means of an example (Section 2). Next, we report on the measurements we have obtained by considering the storage utilization efficiency of the new variation of S-Index, next to the original version as well as to the inverted index (Section 3). Finally, Section 4 concludes this work.

2 The Compressed S-Index Variation

S-Index can be represented by a tree structure. Out of a number of alternative tree-structures considered (trees with different fanouts), the scheme achieving the best performance was found to be the binary tree [6] [6]. Section 2.1 describes the construction of the tree and the contents of its nodes for S-Index. Section 2.2 explains how all the text blocks containing a search word can be retrieved with the help of S-Index. A detailed example is provided in Section 2.3.

2.1 Signature Insertion

In the S-Index schema, each distinct, non-common word is mapped to a unique number in [0,...,V-1], where V is the total number of distinct words in the textbase, through the use of a B+ tree or a MPHF. The textbase is then divided into logical blocks, each containing D distinct words. For every block an ERSF type signature is created. The size of a signature is $M = 2^{log_2 V}$ (the first power of 2 which is equal to or greater than V). This means that a signature has its bits numbered in the [0,...,V-1,V,...,M-1] range. Since only D of the bits [0,...,V-1] can be set to 1 in a signature, bits in [V,...,M-1] are always set to 0 and are reserved for future vocabulary expansion.

Once an ERSF type signature is created for a text block b, it has to be inserted into the S-Index (binary) tree structure. Starting from the root, if the number of 1s in the signature of block b is greater than or equal to the number of 0s, then the signature is stored under the root node, alongside with the corresponding block identifier b, and the algorithm terminates. On the other hand, if the number of 1s is less than the number of 0s, the signature is divided into two equally sized signatures. The first half contains the left M/2 bits and the second half the right M/2 bits of the original signature. The two halves are inserted recursively into the left and right subtrees of the root. The recursive pseudo-code for the insertion of an ERSF type signature of a block in S-Index is shown in Table 2.

Because an S-Index tree is a full binary tree, if V is not a power of 2, then some of the nodes exist but are not used. The full binary tree contains $log_2 M$ levels of nodes, since this is the number of levels needed, so that it is impossible for a signature to be further divided. A block signature may be divided all the way up to the point where two-bit sub-patterns are produced. The only possible combinations for such two-bit patterns are 01 and 10. This is because 00 is not stored and 11 implies that the corresponding signature would have been appended as a four bit pattern, one level up from the current one. By convention, the root level is at level 0 and the leaf level is at level $log_2 M - 1$.

2.2 Single term query processing

Block signatures appended under the root node utilize the whole of the textbase vocabulary. The left child of the root utilizes the left (lower) half of the vocabulary [0,...,M/2-1], because only the left (lower) part of an ERSF signature can be stored under that node. Similarly, the right child of the root registers information on blocks containing words with codes in [M/2,...,V-1], because only the right part of an ERSF signature can be stored there, and so on.

```
PROCEDURE Insert_Signature_Into_Subtree(signature, node)
BEGIN
   IF (size_of_signature < 2 bits long) THEN EXIT this procedure;
   IF (number_of_1s ≥ number_of_0s in the signature) THEN
      append a new record under the current node;
   ELSE
      signature_L = left half of signature;
      signature_R = right half of signature;
      node_L = left child of node;
      node_R = right child of node;
      CALL Insert_Signature_Into_Subtree(signature_L, node_L);
      CALL Insert_Signature_Into_Subtree(signature_R, node_R);
   END
END

CALL Insert_Signature_Into_Subtree(ERSF_Type_Signature_For_Block, root);
```

Table 2. Pseudo-code for inserting a signature into S-Index

When attempting to find all the blocks that qualify for a given single term query, one has to start from the root node and follow a single path down to the leaf level. Along this path, all signatures under each one of the nodes encountered are fetched and examined. If the appropriate bit is set to 1, then the block number is retrieved. The block numbers retrieved are then mapped to addresses on the disk, with the help of an address table constructed during the partitioning of the textbase into logical blocks. The actual text may then be fetched from the disk, by following the corresponding address pointers.

2.3 Detailed Example

As an example, let us consider the following text: **"This is an example for a small text database with common words. Common words in the text are not indexed."** The size of the vocabulary in this textbase is V=7, so M=8 (1 bit is wasted). The mapping of the words indexed is presented below.

example	0
small	1
text	2
database	3
common	4
words	5
indexed	6

The textbase is divided into logical blocks, each containing D=3 distinct words:

Block number	**Block context**	**Block signature**
0	This is an example for a small text	$S_0 = 11100000$
1	database with common words.	$S_1 = 00011100$
2	Common words in the text	$S_2 = 00101100$
3	are not indexed	$S_3 = 00000010$

The empty and populated S-Index structures appear in Figure 2 and Figure 3.

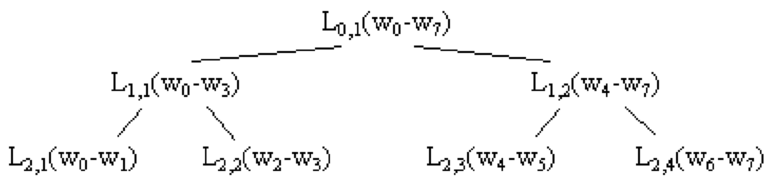

Fig. 2. Empty S-Index structure.

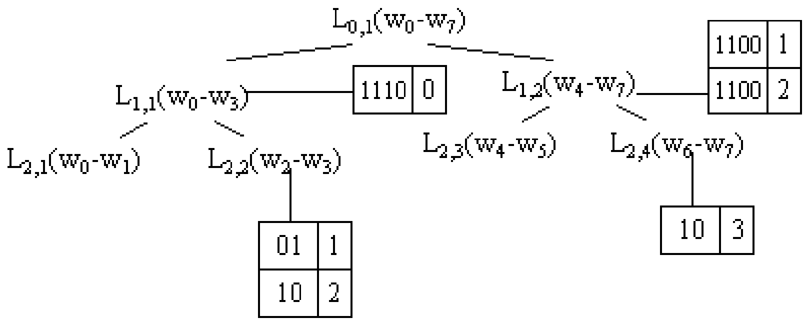

Fig. 3. Populated S-Index structure.

Let's see how block signature S_1 is inserted into the index. S_1 cannot be placed under the root because it has three 1s and five 0s. Thus, it is divided into two signatures S_{1_L} (0001), and S_{1_R} (1100). S_{1_L} cannot be placed under node $L_{1,1}$ because it contains one 1 and three 0s. S_{1_L} is therefore divided into $S_{1\_L\_L}$ (00) and $S_{1\_L\_R}$ (01). $S_{1\_L\_L}$ is not stored under node $L_{2,1}$ because it contains no 1s. $S_{1\_L\_R}$ is stored under node $L_{2,2}$, alongside with its block identifier, since it contains one 1 and one 0. In a similar way, S_{1_R} and block number 1 are appended to the list under node $L_{1,2}$, because half of the signature bits are set to 1.

In the course of processing the single term query "find all the occurrences of the word text in the textbase", first the word text has to be mapped to its corresponding code, which is number 2. This is achieved with the help of a B+ tree or a MPHF.

Next, we consider the root of the S-Index tree. For the example in question, no records are stored under the root node, thus the search algorithm chooses a child node to continue. The latter is node $L_{1,1}$ and is determined by considering the word code (2, in this case). All of the signature records stored under the given node are retrieved and checked. When the appropriate bit position (here, bit position 3) is set to 1, the corresponding block identifier is retrieved. In this case, block number 0 is retrieved. The next child node to be considered is $L_{2,2}$. Bit position 1 of all the signatures stored under this node is checked. The algorithm retrieves block number 2 from node $L_{2,2}$ and terminates because a leaf node has been reached. It is now clear that the word text appears in blocks 0 and 2. This can be confirmed by scanning the corresponding textbase blocks.

3 Storage Utilization Efficiency

In this section we examine the strategy we followed in order to implement S-Index, the implementation environment and the results we obtained.

3.1 Implementation strategy

This subsection covers the implementation details for S-Index. As shown in Figure 4, S-Index consists of two main parts: (a) the skeleton which is a full binary tree containing the nodes, and (b) under each node a list of records, each being of the (block identifier, block signature) type. In the following we describe the way this structure is organized on disk.

Root level (level 0). All records under the root node are stored in a linked list on the disk. A typical root record stores a signature of size M and a block number. The signature utilizes $\lceil M/8 \rceil$ bytes on the disk and the text block number is stored as a two-byte integer. A pointer to the first record of the root's record list on the disk is stored in the first row of a look-up table containing $log_2 M$ rows. A traversal of the list retrieves the remaining records.

Intermediate level (level i, where $0 < i < log_2 M - 1$). All records, in all lists, under all nodes in level i, are stored on the disk in a combined linked list - a structure that consists of many individual linked lists. A typical record, in the list under any node at level i, consists of a $M/2^i$ bit signature, a text block number and a pointer to the next record in the list. The signature utilizes $\lceil M/2^{i+3} \rceil$ bytes on the disk. In addition, two bytes are used for the block number and four are reserved for the next record pointer.

The combined linked list is constructed with the help of the following algorithm: At first, space for 2^i records is allocated sequentially on the disk. Each one of these empty records is the head record to the list of each of the 2^i nodes in level i, and initially has the next record pointer and the text block identifier field set to NULL. In the $(i+1)_{th}$ row of the lookup table, a pointer to the first record of the combined linked list is stored. Once the head record of a list has been used, new records are appended right after it.

In the example below we demonstrate the implementation of the combined linked list for the third level (level 2) of an S-Index tree having M = 16. The size of a signature for level 2 is $\lceil M/2^i \rceil = \lceil 16/2^2 \rceil = 4$ bits. Figure 4 illustrates the four nodes at level 2, and the corresponding combined linked list of records.

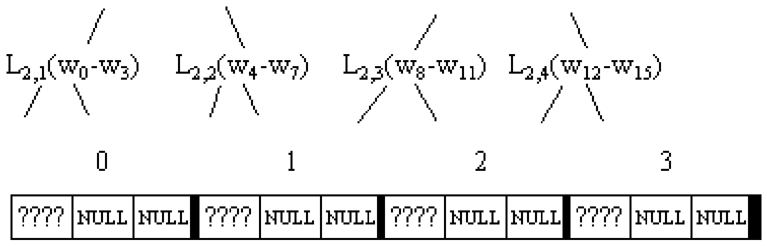

Fig. 4. The empty structure and the combined linked list for level 2 of an S-Index when M = 16.

Record 0 is the empty head record for node $L_{2,1}$, record 1 is the empty head record for node $L_{2,2}$ and so on. Space for four empty head records has been allocated serially on the disk.

After four insertions, the S-Index schema and the corresponding combined linked list at level 2 are shown in Figure 5.

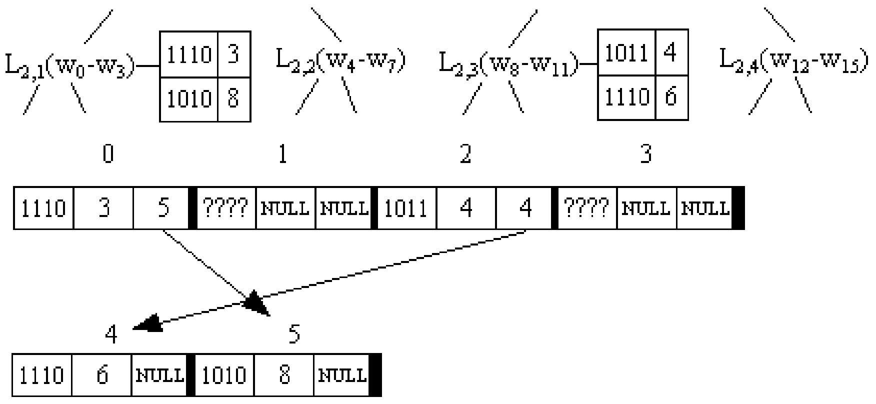

Fig. 5. The tree structure and the combined linked list after four record insertions.

Suppose now that record (1110,9) is to be inserted into the list under node $L_{2,3}$. Figure 6 illustrates the updated tree and combined linked list structures for this case.

In order to traverse the entire list under node j of level i ($L_{i,j}$ where $0 < i < log_2 M - 1$ and $1 \leq j \leq 2^i$) we should: (a) go to the combined linked list with the help of the $(i+1)_{th}$ row of the look-up table, (b) jump to the j_{th} record of the combined linked list, which is the head record to the list under node $L_{i,j}$, and (c) retrieve the whole list by following the next record pointer, until a NULL value is encountered.

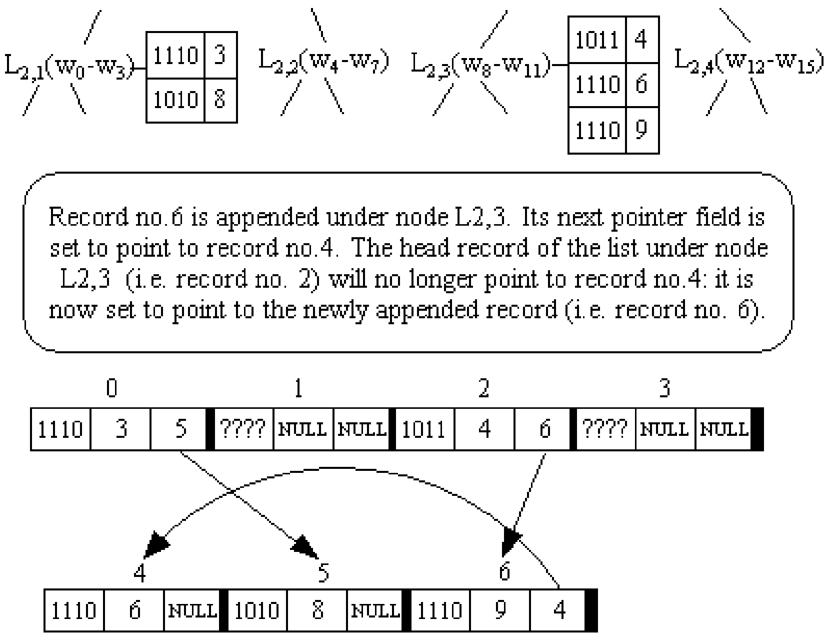
L2,1(w0-w3)
1110 3
1010 8
L2,2(w4-w7)
L2,3(w8-w11)
1011 4
1110 6
1110 9
L2,4(w12-w15)
Record no.6 is appended under node L2,3. Its next pointer field is set to point to record no.4. The head record of the list under node L2,3 (i.e. record no. 2) will no longer point to record no.4: it is now set to point to the newly appended record (i.e. record no. 6).
0 1 2 3
1110 3 5 ???? NULL NULL 1011 4 6 ???? NULL NULL
4 5 6
1110 6 NULL 1010 8 NULL 1110 9 4

Fig. 6. The tree and the combined linked list after the insertion of (1110, 9) under L2,3.

Leaf level (level $log_2M - 1$). The leaf level is stored with the help of a combined linked list, like any intermediate level. The differentiation has to do with the signatures it registers. In the case of S-Index, the possible combinations for the two bit signatures in the leaf nodes are 01 and 10. Thus, for each one node we maintain two record lists, one for each one of the two (possible) 2-bit signature patterns. Evidently, each one record of the two lists in question stores a (block identifier, next record pointer) pair.

From the discussion above, it follows that the binary tree structure (skeleton) is neither saved on the disk nor is maintained in main memory. The only structure that needs to be stored is the look-up table that directs to the combined linked lists and allows for efficient retrieval of the records stored under the S-Index nodes.

3.2 Implementation environment

We developed S-Index and conducted experiments on a single processor (Intel Pentium III 866MHz) PC with 256 megabytes of main memory. The operating system was Linux. For an I-Index environment, we used the MG system[1] [18] [18], which utilizes

[1] MG is available via ftp from ftp://munnari.oz.au/pub/mg.

compressed inverted files for indexing textbases. The textbases used for the experiments are document collections that were generated with FINNEGAN[2] [18] [18]. Their profile is outlined in Table 3. FINNEGAN is a textbase generator that can create synthetic textbases of any size, with statistical properties analogous to those of real text. The 1.2 MB textbase is used to compare our implementation of S-Index with a simulation of the original variation that was conducted on the same textbase [4] [4]. The 130 MB collection is used to compare our implementation of S-Index with I-Index, focusing on storage utilization.

Collection Size	1.2 MB	130 MB
Filter Size (number of common words)	434	598
Vocabulary Size (number of distinct words)	15922	128727

Table 3. The textbase profile

In our implementation of S-Index, the mapping of the distinct words to unique code numbers is achieved with the help of a B+ tree. The numbers are assigned to the words sequentially. This means, that the number code assigned to the first distinct word of the textbase is 0, the number assigned to the second distinct word is 1, and so on.

3.3 Experimental results

Figure 7 presents the results obtained by considering several blocking factor (D) values for S-Index, and includes the curve of the compressed I-Index scheme [18] [18]. The size of S-Index decreases as the value of the blocking factor (D) increases. For a particular range of D values, S-Index requires much less space on disk, than I-Index. For the textbase considered, for D = 4500 then S-Index and I-Index produced indexes of the same size. Nevertheless, upon increasing the value of D, S-Index produced more compact indexes. For D = 12000, S-Index was measured to be only 4.28% the size of the indexed textbase. This implies an improvement of about 57% over the I-Index.

Finally, we compared our implementation of S-Index with the originally proposed S-Index. We used the same textbase (a 1.2 MB collection created by FINNEGAN) as in [4] [4]. Figure 8, shows that the implementation of S-Index proposed in this paper produces smaller indexes.

4 Conclusion

In this paper we presented in detail the schema and implementation of an improved variation of S-Index. The storage utilization efficiency of S-Index was considered next to a 130 MB textbase. The experimental results obtained indicate that the improved variation may be configured to utilize an index size that is less than 5% the size of the

[2] FINNEGAN is available via ftp from ftp://munnari.oz.au/pub/finnegan.

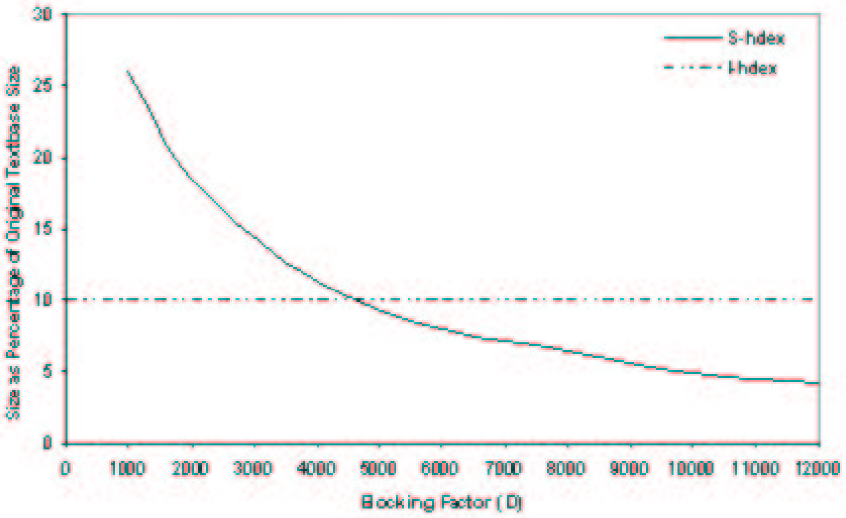

Fig. 7. Disk space requirements for S-Index and I-Index.

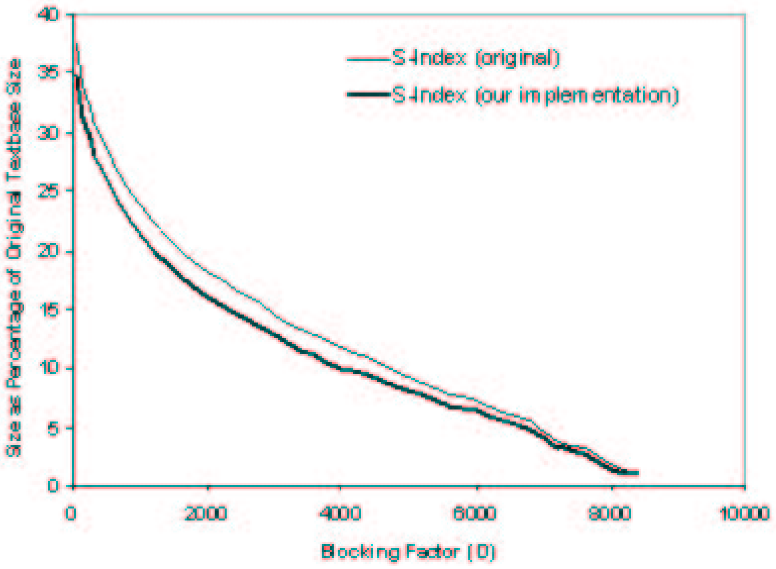

Fig. 8. Disk space requirements for the two variations of S-Index.

indexed textbase corpus. This in turn implies performance comparable to that of the compressed inverted index.

This study emphasizes on measuring the performance of S-Index with regard to storage utilization efficiency. In the future stages of our research, we intent to focus on the S-Index query processing efficiency. It is worth noting that the scheme allows for further improvement with regard to storage utilization efficiency: the list of records stored under the leaf nodes are of the same type as those of the inverted index. In this respect, S-Index is expected to benefit from implementing compression, analogous to that of the self-indexing inverted files, reported to reduce the size of each one inverted list identifier record down to one byte [14] [14].

References

1. Adjeroh D.A. and Nwosu K.C., "Multimedia Database Management - Requirements and Issues" *IEEE Multimedia*, Vol.4, No.3, pp.24-33, Sept. 1997.
2. Blumer A., Blumer J., Haussler D. and McConnell R., "Completed Inverted Files for Efficient Text Retrieval and Analysis", *Journal of the ACM*, Vol.34, pp.578-595, 1987.
3. Croft W.B. and Turtle H.R., "Retrieval Strategies for Hypertext", *Information Processing and Management*, Vol.29, No.3, pp.313-324, 1993.
4. Dervos D., "Binary Signature Schemes for Textbases", Ph.D. thesis (in Greek), Computer Science Dept., Aristotle University, Thessaloniki, Greece, January 2000. Available: http://www.aetos.it.teithe.gr/ dad.
5. Dervos D., Linardis P. and Manolopoulos Y., "Perfect Encoding: a Signature Method for Text Retrieval", *Third International Workshop on Advances in Databases and Information Systems (ADBIS'96)*, ACM Moscow Chapter, Sept.10-13, 1996.
6. Dervos D., Linardis P. and Manolopoulos Y., "S-Index: A Hybrid Structure for Text Retrieval", *1st East European Symposium on the Advances in DataBases and Information Systems (ADBIS'97)*, pp. 204-209, St. Petersburg, Sept. 1-5, 1997.
7. Elmasri R. and Navathe S.B., *Fundamentals of Database* Systems, Benjamin/Cummings, Menlo Park, California, 1989.
8. Faloutsos C., "Signature Files", in *Information Retrieval Data Structures and Algorithms*, Frakes W.B. and Baeza-Yates R. (Eds.), pp.44-65, Prentice Hall, Englewood Cliffs, N.J., 1992.
9. Fox E.A., Chen Q.F. and Heath L.S.: "A Faster Algorithm for Constructing Minimal Perfect Hash Functions", *Proc. ACM SIGIR Conference*, pp.266-273, Copenhagen, June 1992.
10. Fox E.A., Chen Q.F., DAOUD A.M. and Heath L.S.: "Order-Preserving Minimal Perfect Hash Functions and Information Retrieval", *ACM Transactions on Information Systems*, Vol.9, No.3, pp.281-308, July 1991.
11. Harman D., Fox E., Baeza-Yates R.A. and Lee W., "Inverted Files", *in Information Retrieval: Data Structures and Algorithms*, by Frakes W. and Baeza-Yates R. (Eds.), Prentice Hall, pp.28-43, 1992.
12. IBM, DB2 UDB, Road Map to DB2 v.7.1 Programming, [Online], International Business Machines Corporation, 2000. Available: http://www.software.ibm.com/data/db2/library/.
13. Ishikawa Y., Kitagawa H., Ohbo N., "Evaluation of Signature Files as Set Access Facilities in OODBs", *ACM SIGMOD'93 Proceedings*, pp. 247-256, May 1996.
14. Moffat A. and Zobel J., "Self-Indexing Inverted Files for Fast Text Retrieval", *ACM Transactions on Information Systems*, Vol.14, No.4, pp.349-379, Oct. 1996.
15. Rabitti F. and Savino P., "Image Query Processing Based on Multi-level Signatures", *Proc. ACM SIGIR Conference*, pp.305-314, Chicago, 1991.

16. Sacks-Davis R. and Ramamohanarao K., "A Two Level Superimposed Coding Scheme for Partial Match Retrieval", *Information Systems*, Vol.8, No.4, pp.273-280, 1983.
17. Turtle H.R. and Croft W.B., "A Comparison of Text Retrieval Models", *The Computer Journal*, Vol.35, No.3, pp.279-290, 1992.
18. Zobel J., Moffat A. and Ramamohanarao K., "Inverted Files Versus Signature Files for Text Indexing", *ACM Transactions on Database Systems*, Vol.23, No.4, pp.453-490, Dec. 1998.