Applying SD-Tree for Object-Oriented Query Processing

I. Elizabeth Shanthi
Dept of Computer Science
Avinashilingam University for Women, Coimbatore, India.
E-mail: shanthianto@yahoo.com

R. Nadarajan
Dept of Mathematics and Computer Applications
PSG College of Technology, Coimbatore, India.
E-mail: nadarajan_psg@yahoo.co.in

Keywords: signature files, B+ tree, indexing, object oriented data bspases

Received: April 14, 2008

We follow signature-based approach to object-oriented query handling in this paper. The use of signature files as an index for full text search has been widely known and used. Signature file based access methods initially applied on text have now been used to handle set-oriented queries in Object-Oriented Data Bases (OODB). All the proposed methods use either efficient search method or tree based intermediate data structure to filter data objects matching the query. Use of search techniques retrieves the objects by sequentially comparing the positions of 1s in it. Such methods take longer retrieval time. On the other hand tree based structures traverse multiple paths making comparison process tedious. In this paper we describe a new indexing technique for representing signature file using the dynamic balancing of B+ tree called Signature Declustering tree (SD-tree). The structure has the positions of 1s in the signatures distributed over a set of leaf nodes. Using this for a given query signature all the matching signatures can be retrieved cumulatively from a single node. Also for signature insertion an optimal search path is calculated by keeping a threshold value and by using forward pointers in leaf nodes. To promote optimal search between subsequent queries the backward leaf node pointers are used. Experiments have been conducted to analyze the time and space overhead of the SD-tree by varying the signature length and the distribution of signature weight for varying query signature patterns. Also, to validate the proposed structure a hypothetical object schema is considered and sample queries evaluated.

Povzetek: SD drevesa so uporabljena za objektno obravnavo vprašanj.

1 Introduction

The advent of internet has made the volume of data going high everyday in all computer-based applications. This has entrusted researchers to design more powerful techniques to generate and manipulate large amounts of data to derive useful information. Indexing plays a vital role in the fast recovery of required data from large databases.

Among the many indexing techniques reported the signature file approach is preferred for its efficient evaluation of set-oriented queries and easy handling of insert and update operations. Initially applied on text data [2, 3, 12, 17, 21] it has now been used in other applications like office filing[6], relational and Object-Oriented Databases[14,16,26] and hypertext[9].

Signatures are hash coded abstractions of the original data. It is a binary pattern of predefined length with fixed number of 1s. The attributes' signatures are superimposed to form object's signature. To resolve a query, the query signature say Sq is generated using the same hash function and compared with signatures in the signature file for 1s sequentially and many non-qualifying objects are immediately rejected.

If all the 1s of Sq matches with that of the signature in the file it is called a drop. The signature file method guarantees that all qualifying objects will pass through the filtering mechanism; however some non-qualifying objects may also pass the signature test. The drop that actually matches the Sq is called an actual drop and drop that fails the test is called false drop. The next step in the query processing is the false drop resolution. To remove false drops each drop is accessed and checked individually. The number of false drops can be statistically controlled by careful design of the signature extraction method [7] and by using long signatures [3,6].

1.1 Related work

Different approaches have been discussed by researchers to represent Signature file in a way conducive for evaluating queries, such as Sequential Signature File[31], Bit-Slice Signature file[31], Multilevel Signature file[25], Compressed Multi Framed Signature file[23], Parallel Signature file[20], S-Tree and its variants[13,24], Signature Graph[28] and Signature tree[27,29,30].

1.2 Motivation for the current work

The signature tree developed by Chen [30] is having the following drawbacks:

- Signatures are inserted considering both 0s and 1s whereas actual weight age is for set bits only.
- Insertion path is dictated by the existing tree structure.
- To process a query, bits appearing in the tree from root node are compared with query signature pattern for 0s and 1s and not by its set bits.
- For a 0-bit in the query both left and right sub tree is followed leading to multiple traversals.

These observations laid the foundation for the current work. We study a new indexing technique for OODBSs using the dynamic balancing of B+ tree called Signature Declustering (SD)-tree in which the positions of 1s in the signatures are distributed over a set of leaf nodes. Using this for a given query signature all the matching signatures can be retrieved cumulatively in a single node.

The rest of the paper is organized as follows. In section 2 we discuss briefly the different approaches used to represent the signature file. In section 3 the structure of the proposed SD-tree is shown. Section 4 is devoted to the algorithms for insert, search and delete operations. Section 5 proposes a sample data set and queries to validate the new structure. Section 6 reports the results of the experiments conducted with the analytical comparison of SD-tree with that of Signature tree [30]. Finally section 7 concludes the work with further outlook on the work.

2 A summary of signature file techniques

2.1 Signature files

A Signature is a bit string formed from a given value. Compared to other index structures, signature file is more efficient in handling new insertions and queries on parts of words. Other advantages include its simple implementation and the ability to support a growing file. But it introduces information loss which can be minimized by carefully selecting the signature extraction method.

Techniques for signature extraction such as Word Signature (WS) [2,3,4,6], Superimposed Coding (SC) [1,2,3,4,5,6,10], Multilevel Superimposed Coding [12], Run Length Encoding (RL) [3,4,6], Bit-block Compression (BC) [3,4], Variable Bit-block Compression (VBC) [4,6] have been reported. The encoding scheme sets a constant number say m, of 1s in the range [1..F], where F is the length of the signature. The resulting binary pattern with m number of 1s and (F-m) number of 0s is called a word signature. The signature of a text block or object can be obtained by superimposing (logical OR operation) all its constituent signatures (i.e) word signatures for text block and attributes' signatures for object. The set of all signatures form a signature file. An example of Superimposed Coding and a sample query evaluation is given below.

Information Retrieval	0010 0100 0100 0001
Block Signature	0110 0101

Sample queries

Matching query
    Keyword = Information   0010 0100
    Query descriptor   0010 0100
Block signature matches   (Actual Drop)

False Match query
    Keyword = Coding   0010 0001
    Query descriptor   0010 0001
Block signature matches   (False Drop)
    but keyword does not

Non-matching query
    Keyword = Information   0010 0100
    Keyword = Science   0000 0110
    Query descriptor   0010 0110
Block signature does not match

2.2 Applications of signatures

Signatures are applied mainly in database access methods useful for text retrieval such as Full text scanning, Inversion, Clustering, Multi attribute retrieval methods like hashing and signature files. Such applications are discussed in [3,5,21]. Here the documents are stored sequentially in the "Text File".

Signatures which are abstractions of the documents are stored in the "Signature File". The latter serves as a filter on retrieval. It helps in discarding a large number of non-qualifying documents. Signatures have been applied in areas rich in text documents like telephone directory [1], office systems [3,4,6], Optical and Magnetic disk access [8], Data base Management system and Library automation.

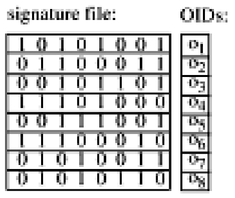
Other applications include
Access method for documents
Indexing method for large text file [17, 18].
Access method for formatted data[1].
To speed up searching in editor[7].
To compress a vocabulary[7].
For a spelling checking program[7].
In differential file[7].

2.3 Physical representations of signature files

This section discusses briefly the various techniques used to represent signature files. We follow the lead of [30] for figures in this section.

2.3.1 Sequential Signature File (SSF)

The signatures are sequentially stored in a file called SSF as in Fig. 1. In single level signature methods every signature must be accessed and tested. Since signatures are abstractions of original data with smaller size, the method is faster than sequential scan of objects themselves. This method is easy to implement and requires low storage space and low update cost. The disadvantage is that more the number of objects exist, the more is the time spent on scanning signature file [8,31]. Therefore it is generally slow in retrieval. To support faster access multilevel signature methods are suggested.

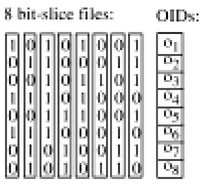


signature file:								OIDs:
1	0	1	0	1	0	0	1	o_1
0	1	1	0	0	0	1	1	o_2
0	0	1	0	1	1	0	1	o_3
1	1	1	0	1	0	0	0	o_4
0	0	1	1	1	0	0	1	o_5
1	1	1	0	0	0	1	0	o_6
0	1	0	1	0	0	1	1	o_7
0	1	0	1	0	1	1	0	o_8

Figure 1: A typical Sequential signature file.

2.3.2 Bit-Sliced Signature File (BSSF)

BSSF stores signatures in a column-wise manner as illustrated in Fig. 2. Thus F bit-slice files one for each bit position of the set signatures are used. In retrieval only a part of the F bit-slice files have to be scanned and hence the search cost is lower than SSF. However update cost is higher. This is because a new signature insertion requires about F disk accesses one for each Bit-slice file [8,31].



8 bit-slice files:								OIDs:
1	0	1	0	1	0	0	1	o_1
0	1	1	0	0	0	1	1	o_2
0	0	1	0	1	1	0	1	o_3
1	1	1	0	1	0	0	0	o_4
0	0	1	1	1	0	0	1	o_5
1	1	1	0	0	0	1	0	o_6
0	1	0	1	0	0	1	1	o_7
0	1	0	1	0	1	1	0	o_8

Figure 2: A typical Bit-Sliced Signature File.

2.3.3 Compressed Bit-Sliced Signature File (CBSSF)

By choosing a proper hashing function for signature extraction the number of 1s is forced to be one. Here, the signature length should be increased to maintain the false drop probability at minimum. This creates a sparse matrix which is easy to compress [8, 23]. A simple way to compress is to replace each 1 with its corresponding physical address.
In Fig. 3 the hash table has a list of pointers pointing to the heads of linked list [8].

For example assume that the word *Text* has its first bit set to 1 and it appears at the 50th byte of text file then searching the first bucket list, we find the position of the word *Text*. Although this approach gives some space saving, the number of false drops will definitely be increased due to sparse signature files.

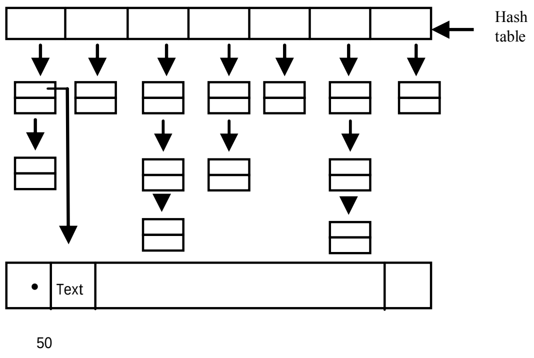


Figure 3: A typical Compressed Bit-Sliced Signature File organization.

2.3.4 S-Tree

S-Tree [24] is a B+ tree like structure [11,22] with leaf nodes containing a set of signatures with their Object Identifiers (OIDs). The internal nodes are formed by superimposing the lower level nodes as shown in Fig. 4. For example to retrieve a query signature Sq = 11000000, we search S-ree top down. From root node v1 we compare and move to v6. In the next level both v7 and v8 match and finally we end up with signatures o11, o12 in v7 and o13 in v8.

The advantage is simple tree searching way of obtaining signatures rather than searching the whole signature file.

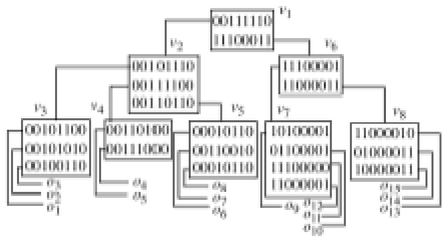


Figure 4: A typical S-tree organization.

The disadvantage is that due to superimposing, internal nodes in the upper level tend to have more weight which ultimately decreases selectivity. The S-Tree has been further improved in [13, 14]. In [13] a number of new split methods namely Linear split, Quadratic split, Cubic split and hierarchical clustering for S-tree is proposed to improve query response time. In [14] a new hybrid scheme combining linear hashing, S-tree and parametric weighted filter is used to evaluate subset-superset queries.

2.3.5 Multilevel Signature file

The structure is similar to S-Tree. However a signature at non-leaf node is formed by superimposed coding from all text blocks indexed by the subtree of which the signature is the root. Fig. 5 shows multilevel signature file for the set of signature values depicted in

4 Fig 4. Though this method improves selectivity in an n internal node, it requires more space. An improved e method for multilevel signature file is discussed in [25].

2.3.6 Signature Graph

The signature file is organized as a trie like structure [28,30]. However, the path visited in the graph to find a signature that matches a given query signature corresponds to a signature identifier which is not a continuous piece of bits, differentiating the signature graph from trie.

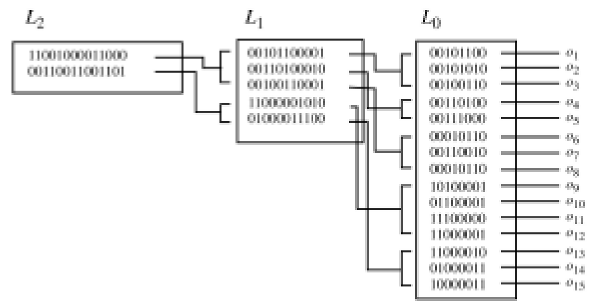


Figure 5.A: Typical Multilevel Signature file organization.

Though signatures are represented compactly, the search path length is not same for all queries. In other words the graph is not balanced. In worst case it degrades to a signature file. Fig. 6. shows the signature file and the corresponding signature graph.

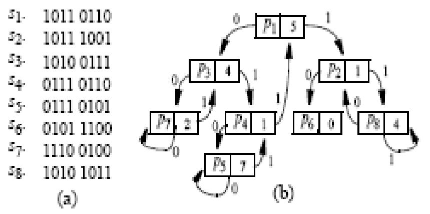


Figure 6: A typical signature file and signature graph.

2.3.7 Signature Tree

The signature tree is a binary tree like structure with nodes representing the bit positions and left and right sub tree followed for binary values 0 and 1 respectively. Each signature is identified from the root by checking the bit positions dictated by the nodes. Nevertheless for a query signature the tree is searched top to bottom according to the bit positions dictated by the nodes rather than the 1s in query signature. Also, for a match with bit 1, searching follows the right sub tree and for 0 at a node both left and right sub trees are followed.

That is for a balanced signature tree more than one path is traversed. Fig. 7. indicates a signature file and its corresponding signature tree. The thick lines in Fig 7(b) indicates the signature identifier that corresponds to S3. In the following section we discuss a quite different method called SD-tree which considers only the positions of 1s in a given signature and decluster them over a set of leaf nodes so that query response time is improved.

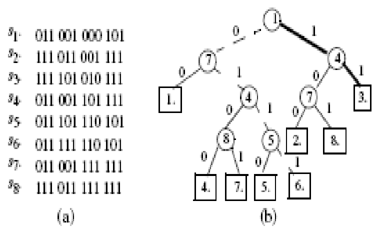


Figure 7: A typical signature file and signature tree.

3 The Structure of SD-tree

In this section we describe the structure of SD tree. There are three types of nodes:

- Internal nodes
- Leaf nodes
- Signature nodes

The internal nodes and leaf nodes are somewhat similar to the internal nodes and leaf nodes of B+ trees respectively. The internal nodes form the upper tree and leaf nodes at last but one level. The signature nodes are at the bottom level of the SD-tree. We will now explain the structure of the nodes in detail. To make discussion simple, we assume the tree order as 3 for a signature file with 8 block signatures of length 12.

3.1 Structure of Internal node

An internal node of SD-tree is illustrated in Fig. 8. Like B+ tree internal node pointers and keys alternate each other. For a tree of order 3 the internal node has two keys K1 and K2 and three pointers P1, P2, P3. These pointers are tree pointers pointing to the nodes at the lower level.

While searching, the left tree pointer is followed for values less than or equal to the node value, else right pointer is followed for values greater than the node value as in B+ tree.

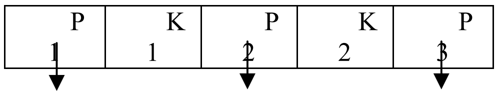


Figure 8: A typical structure of an internal node.

3.2 Structure of a leaf node

The leaf nodes appear in the last but one level of the SD tree. Like B+ tree all the key values appear in ascending order of their values in the leaf nodes and are connected to promote sequential search. But unlike B+ tree in SD-tree each value is followed by a signature node instead of data pointer. This is depicted in Fig 2. Pointers P1 and P2 point to the corresponding signature nodes for K1, K2; P3 is the sequential pointer to next leaf node and P4 is the backward pointer from next leaf node.

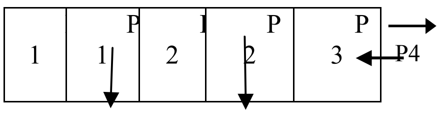


Figure 9: A typical leaf node entry.

3.3 Structure of signature node

The structure of a signature node is shown in Fig. 10. The signature node for Ki has 2^{i-1} binary combinations denoting the possible prefixes. When a signature S_u with 1 in the i th position is to be inserted the intermediate prefix formed (explained in section 4) is compared with the binary combinations in the signature node at Ki and u is inserted in the list.

B1	Signatures having prefix B1
B2	Signatures having prefix B2
...	...
Bn	Signatures having prefix Bn

Figure 10: A typical signature node entry.

3.4 Overall structure of SD-tree

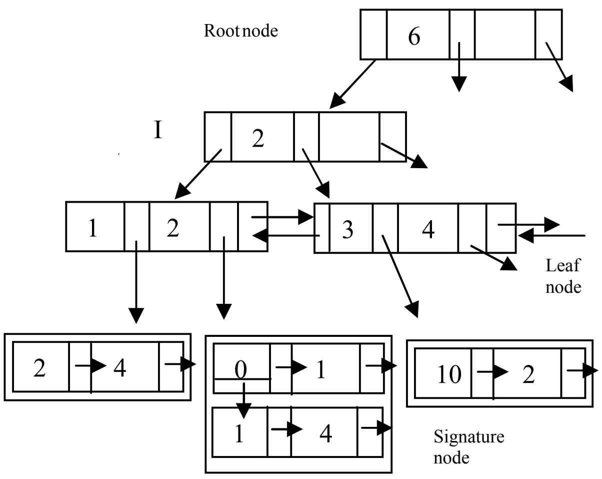
Consider the partially filled SD-tree shown in Fig 11(a) for discussion. The tree has been constructed for signature length F = 12. It is obvious from Fig. 11(a) tree of order 3 has height 2. Consider the signature file in Fig 11(b). To insert signature S1 with first occurrence of 1 at position 2, access the leaf node with value 2, follow its signature node and write the signature value as 1 (for S1) with the prefix 0 (for bit 1). In the same way S2 is inserted in signature node 1 with no prefix and signature node 3 with prefix 10 (for bits 1 and 2). S4 will be inserted at signature node 1 with no prefix and signature node 2 with prefix 1.

While inserting a signature there are two possible options to move to next 1s position in the leaf node. One is via the sequential pointer between leaf nodes and the other is the top-down traversal of tree from root. To ensure optimal search path the threshold (Th) is fixed at h+1 (h being the tree height plus one more level for accessing signature nodes). As long as the number of sequential pointers traversed in leaf nodes is within the specified Th value, follow the sequential pointers.

This is calculated by finding the difference (d) between two consecutive 1s in a signature divided by the number of entries per leaf node. This is given by the condition

$$\frac{d}{p-1} \leq \text{Th} \quad \text{, where p is the order of the tree.}$$

The division here is integer division. When the above condition is not true, a new tree access is initiated from root. Similarly to promote optimal query processing the query signature value Sq is taken in decimal form. The occurrence of the last one (in the least significant position) is found by performing D Mod 2 operation. The remaining binary prefix is formed in the process of decimal–to-binary conversion of D. The procedures for tree maintenance are explained in the following section.



(a)

S1 : 010……….
S2 : 101……….
S3 : 000……….
S4 : 110……….

(b)

Figure 11: Overall structure of SD-tree.

To minimize the number of false drops the database size is selected according to [4, 27, 30] as

$$F \ln 2 = mD \qquad \text{…………………} (1)$$

where F is the signature length, m the number of set bits and D the average data block size.

4 Tree Search and Updates

This section lists the algorithms for signature insert, delete and search operations on SD-tree. A global flag F is set to 0 indicating the search path from the root of the tree by default. In the procedure after the first 1's insertion, depending on the d value F may be set to 1.

4.1 Insertion

The algorithm for signature insertion is outlined in this section. The call to procedure New(node) returns a new link in the signature node.

```
Insert ($S_u$)
  Input : The signature to insert $S_u$ ;
      1. Let $i_1, i_2, ….. i_n$ be the positions of 1 in $S_u$ ;
         F ← 0; Th = h+1; // for tree of order p
      2. Move $i_2$ to $i_n$ to queue Q. B = NULL;
      3. If ($i_1$ = 1) then begin
         Access leaf node $i_1$; // from root
         New(node);
         insert u;
         B = strcat ( B, '1'); // to denote bit 1 position
           end
           else begin
              for k = 1 to $i_1$ – 1 do
              B = strcat ( B, '0');
      Access leaf node $i_1$;   // from root
      New(node); write (B);
      insert u ;
      B = strcat ( B, '1');
    end
    f = $i_1$; // store the current bit position
    F ← 1; // enable sequential search in leaf nodes
4. While Q not empty do
   begin
      read x from Q;
      if (x = f+1) then
      begin
         Access leaf node x;  // via leaf node pointers
         If (not(B)) then  New(node); // create node
    Write(B);
    Write u @ prefix B;
      end
     else begin
            d = x – f;
            If (d/(p-1)) > Th then  F ← 0;
            for k = f +1 to x-1 do
                B = strcat ( B, '0');
            Access leaf node x;
            If (not(B)) then  New(node);
        Write(B);
         Write u @ prefix B
            end
         B = strcat (B, '1');
          f = x;
      end.  // until queue is empty
```

4.2 Searching

The following algorithm outlines the steps to search for signatures matching a given query signature Sq. In the procedure F ← 0 always and the algorithm lands up directly in the signature node corresponding to last 1 from root. The match here is the exact match and the optimal signature length selected in equation (1) minimizes the false drops.

Search(S_q)

Input : The (query) signature to search as decimal D.
Output : The list of signatures matching the given signature.

```
1  Repeat  i = D mod 2; D = D div 2; until i = 1
2  Let n be the corresponding bit position of i.
3  Let B = toBinary(D);  // convert D to binary
4  Access leaf node n  // from root node
5  Search for B.
6  If Found() then read and output the list of
   signatures.
```

4.3 Deletion

The algorithm to delete a signature from SD-tree is described below.

Delete (S_u)

Input : S_u, the signature to delete.

```
1. Let $i_1, i_2, ….. i_n$ be the positions of 1 in $S_u$.
2. For each $i_k$ ($1 \le k \le n$)  form prefix B as in Insert
   ($S_u$).
3. Access the leaf node and follow the signature
   node;
4. Access prefix B and search for u.
5. If present, delete it .
6. Repeat steps (2) through (5) for all $i_k$s.
```

5 A sample validation model

This section discusses the evaluation of sample Object-Oriented queries on a hypothetical object base. Fig 12 shows the class diagrams in UML notation [15,19]. The classes and their relationships are listed in Table 1.

S. No	Class 1	Class 2	Relationship
1.	University	Dept	Composition
2.	University	Student	Aggregation
3.	Student	Programme	Association
4.	Dept	Instructor	Association
5.	Student	Male	Generalization
6.	Student	Female	Generalization
7.	Programme	Subject	Association

Table 1: Class relationships

The sample hashing outputs of attributes values are listed in Table 2.

Dept-name	Pgm-name
Mathematics – 1010 0000	M.E – 0001 0100
Computerscience–0100 1000	M.Sc (S.E) – 1000 1000
Physics – 0001 0010	M.Sc (Mat) – 0100 1000
	M.sc (Phy) – 0011 0000

Inst-name	Stud-name	Male
John – 1000 0010	David – 0100 0010	Sex – 0010000
Adams – 1000 1000	Elena – 1001 0000	Female
James – 0000 1001	Maria – 0100 0001	Sex –00001001
Janes – 0110 0000	Peter – 0010 0001	
	Grace – 0000 1100	
	Antony – 0000 0011	

Table 2: Sample attributes

The attributes' hash coded values are superimposed to produce object's signature. The set of all object signatures of a class correspond to the signature file. The signature files created for various attribute combinations of objects are listed in Table 3.

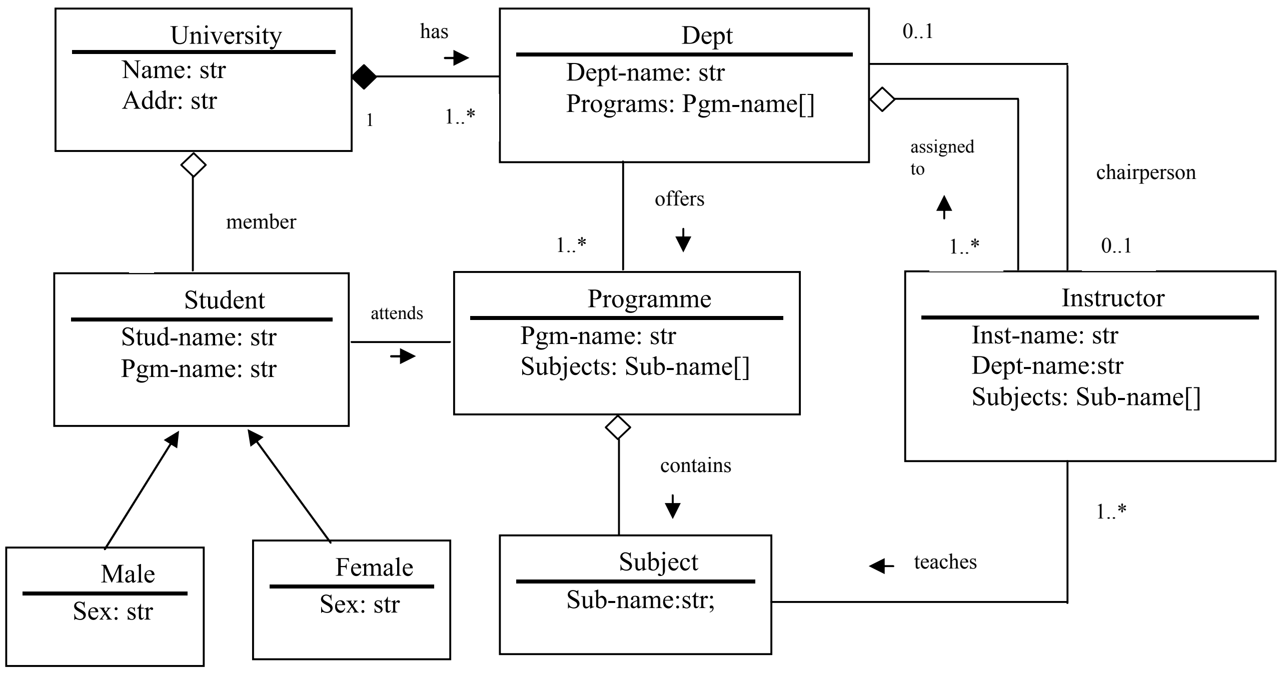


Table 3: Signature files of classes

Class : Subject

Sub-name
1. Software engg – 0100 0001
2. Comp. applns – 0010 1000
3. Comp. engg – 0100 0100
4. Applied Maths – 1000 0001
5. Calculus – 0010 0100
6. Nuclear physics – 0101 0000

Class : Programme

Pgm-name	Subjects	Obj. signature
1. M.E	Comp.applns, Applied Maths	1011 1101
2. M.Sc (S.E)	Software engg, Comp. engg	1100 1101
3. M.Sc(Phy)	Comp. applns, Nuclear physics	0111 1000

Class : Male

Stud-name	Pgm-name	Sex	Object signature
1. David	M.Sc (S.E)	Male	1101 1010
2. Peter	M.E	Male	1011 0101
3. Antony	M.E	Male	1001 0111

Class : Instructor

Inst-name	Dept-name	Subjects	Obj signature
1. John	Comp.sc	Software engg, Comp. applns	1110 1011
2. Adams	Mathematics	Applied Mathematics, calculus	1010 1101
3. James	Comp.sc	Software engg	0100 1001
4. Janes	Physics	Nuclear physics	0111 0010

Class : Female

Stud-name	Pgm-name	Sex	Obj signature
1. Elena	M.Sc (S.E)	Female	1001 1001
2. Maria	M.Sc (S.E)	Female	1100 1001
3. Grace	M.Sc (Phy)	Female	0011 1101

Class : Dept

Dept-name	Programs	Obj signature
1.Computer science	M.E,M.Sc (S.E)	1101 1100
2. Mathematics	M.Sc (Mat)	1110 1000
3. Physics	M.Sc(Phy)	0011 0010

Figure 12: Sample Object schema.

Class : Student

Stud-name	Pgm-name	Obj signature
1. David	M.Sc(S.E)	1100 1010
2. Peter	M.E	0011 0101
3. Antony	M.E	0001 0111
4. Elena	M.Sc(S.E)	1001 1000
5. Maria	M.Sc(S.E)	1100 1001
6. Grace	M.Sc(Phy)	0011 1100

1.	List of students doing a given programme	Pgm-name = M.Sc(S.E)	Student	1000 1000	1100 1010 (David) 1001 1000 (Elena) 1100 1001 (Maria)
2.	Instructors handling a given subject	M.Sc(S.E) in subjects	Instructor	0100 0001	1110 1011 (John) 0100 1001 (James)
3.	Dept(s) offering a given subject	1. Comp. applns in subjects	Programme	0010 1000	1011 1101 (M.E) 0111 1000 (M.ScPhy)
		2. Pgm-name = 00010100, 00110000	Dept		1101 1100 (Comp.sc) 0011 0010 (Phy)
3.	All instructors of a given dept	Dept-name = Comp.sc	Instructor	0100 1000	1110 1011 (John) 0100 1001 (James)
4.	Female students attending a given programme	Pgm-name = M.Sc(S.E)	Female	1000 1000	1001 1001 (Elena) 1100 1001 (Maria)

Table 4: Sample queries

6 Experimental results

To validate the proposed structure we implement SD-tree in Java and for every test run the tree is constructed statically before signature insertion. The parameters considered in the experiments' data sets are Signature length (F), Signature weight (m) and signature weight distribution (swd).

The experiments were carried out in a standalone system with Intel Pentium IV processor. The main memory size is 512 MB and the hard disk capacity is 80 GB.

6.1 Signature tree Versus SD-tree

In this section the parameters which are generally considered in the analysis of indexing structures like time and space complexities are reported. We compare the results of Y. Chen's Signature tree [30] with that of the SD-tree. The observed results are listed in Table 5.

6.2 Time Complexity

Like in other signature applications we use the response time as the performance measure [23]. The time complexity of the insert algorithm initially is the sum of time taken to construct the B+ tree of order p and the time taken for inserting. Here B+ tree is constructed with values 1,2, ….. , F where F is the length of the signature for a given dataset. Hence compared to the use of B+ tree as index structure for large datasets the value F is small which reduces the time taken for SD-tree construction considerably. In algorithm 4.1 the time complexity for insertion is bounded by O(nm) where n is the number of signatures in the file and m is the number of 1s in the given signature as against the O(nF) in the signature tree insertion[30], n being the number of signatures in the file and F is the full length of signature including 0's and 1's. Since deletion follows similar steps as insertion the time complexity is same for both. Another desirable characteristic of SD-tree is that for higher F values, by varying p, the value of h, height of the tree can be kept small to promote faster search. It is observed that the time taken for tree construction when F = 10 and p = 3 is 2015421 nano seconds.

Parameter	Signature tree	SD-tree	Inference
Time complexity	O(nF)	O(nm)	m < F; Faster insertion
Tree height	$O(\log_2 n)$	$O(\log_p (F/(p-1)))$	p>2 ; Shorter tree
Search cost	$O(\lambda.\log_2 n)$	$O(\log_p F+a)$	F < n; Cost < Sig. tree
Space complexity	$n\log_2 F + 2\sum_{i=0 \text{ to } k} 2^i (i+1)$	O(F(a+f))	< Sig. tree when k>F

Table 5: Signature tree Vs SD-tree

In Table 5,
n - Number of signatures in signature file
F – Length of signature
m - Number of set bits
p – Order of SD-tree
λ – Number of path traversed in query searching
a – Average no. of signatures / signature node
k – log 2 n
f - Average no. of prefix values / signature node

Similarly search time is the sum of time taken to access the leaf node (Tl) and signature node search time (Tsi). This is given by

$$Ts = Tl + Tsi$$

Here, Tl is constant for all leaf nodes for a dynamic balanced structure like SD- tree and Tsi is directly dependent on the i value and the number of signatures inserted in the signature node. In the worst case the search time is bounded by $O(Tl + 2^{i-1})$.

To analyze the query response time the signature weight distribution was fixed as 100% , 70%, 50% and 30% for signature lengths 10 and 30. The values are plotted in Fig. 13. through Fig. 16. The weight of the signature was biased in upper byte(U), lower byte(L) or uniformly distributed(M) and time values noted.

For 100% swd the insertion and search time is a constant of the signature weight bias. This is depicted in Fig. 13. In the same way the swd was fixed at 70, 50 and 30 and the observed values are plotted in Fig. 14, Fig. 15 and Fig. 16 respectively. The query response time between two consecutive queries is minimized by following the backward pointers in leaf nodes when the following condition is true. That is,

$$\frac{d_{i,j}}{p-1} \leq Th$$

where $d_{i,j}$ is the difference in Sqi's last 1's position and Sqj's first 1's position.

All the graphs show that the time taken for signature insertion grows linearly with the values of swd, F and the weight bias. Insertion time increases in upper nodes due to the complexity of the circuit in creating prefixes.

SD-tree maintenance and space overhead

SD-tree maintenance is quite simple that the tree is not subject to extensive node split or merge. This is because the insertions and deletions do not affect the node values or the height of the tree. Operations are reflected only in the signature node. In the experiments the binary prefix pattern nodes are created dynamically. The space consumption for insertion of a signature number at a signature node depends on the binary prefix length (l), the pointer size (p) and the space for writing the signature number(n). The prefix is created anew each time if it does not exist. Hence, the space complexity to insert with prefix for a signature of weight (w) is given by O[w((l+2p)+(n+p))]. Similarly the space complexity of a signature for which prefix already exists is bounded by O[w(n+p)].

Fig. 17 shows the space overhead of SD-tree. The tree is created statically. Signature nodes are created dynamically and are of fixed size. It is clear from the graph that the space consumption increases linearly with F and w. It is obvious that for all combinations of values the query search time for SD-tree is lesser than signature insertion time.

For swd of 70%, 50% and 30% the signature weight was biased in lower byte, upper byte or uniformly distributed and values noted. All the outputs clearly indicate that the time taken for signature insertion and query response is slightly higher in upper levels.

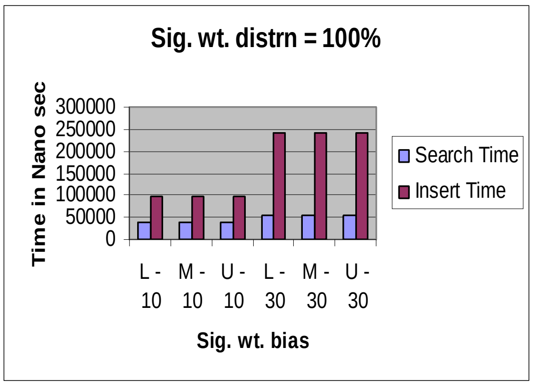


Figure 13: 100% swd.

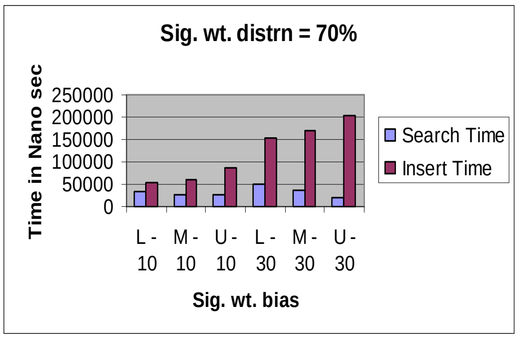


Figure 14: 70% swd.

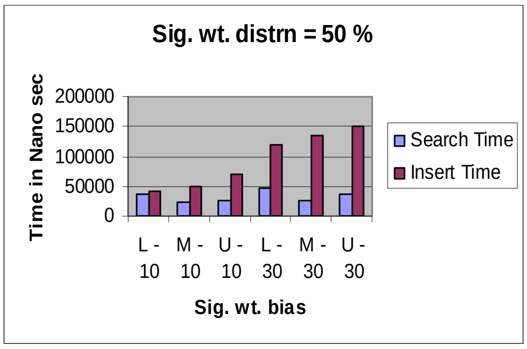


Figure 15: 50% swd.

Nevertheless the query response time is lesser than signature insertion time. As the structure complexity increases in signature nodes in upper levels the swd was analyzed for both ends separately.

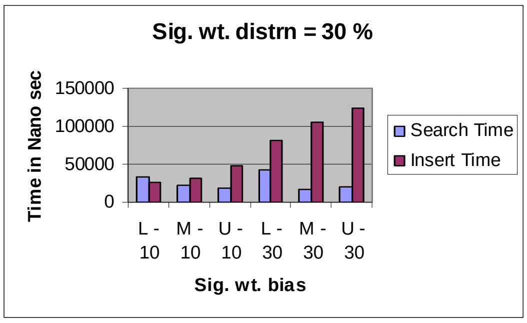


Figure 16: 30% swd.

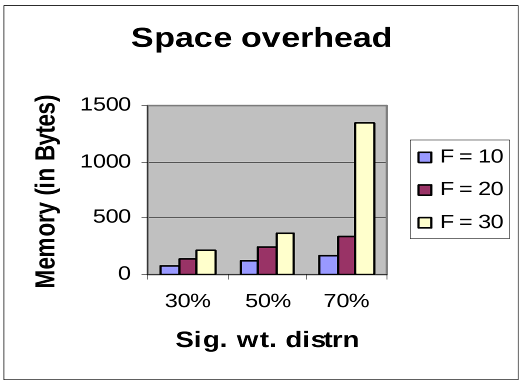


Figure 17: Space overhead of SD-tree.

7 Conclusion and research directions

In this paper we presented a novel way to represent signatures in a B+ tree like structure called SD-tree and analyzed the performance for query response time.

By varying the signature length and distribution of 1s in the signature the query response time was noted and results plotted. It is clear from the graphs that considerable search time is saved.

The space overhead in SD-tree may be higher due to the presence of binary prefixes in higher order signature nodes, but the flexibility provided by the SD-tree outweighs all besides simple maintenance and faster query retrieval time.

The work is proposed to extend in the following directions. The synthetic data sets are to be replaced with, run and verified on a real Object Oriented Data Base system. Another direction is when the signature weight is more than 50%, use 0s so that number of signature nodes accessed for insertion and search is optimal. Also the structure can be modified to support point and range queries in Object Oriented Data Base system.

References

[1] Charles S. Roberts, (1979) *Partial-Match Retrieval via the Method of Superimposed Codes*, Proc. of IEEE, Vol. 67, No. 12, pp 1624 – 1642.

[2] Chris Faloutsos, Stavros Christodoulakis, (1984) *Signature Files: An Access Method for Documents and Its Analytical Performance Evaluation*, ACM Trans on Office Information Systems, Vol.2, No. 4, pp 267 – 288.

[3] Christos Faloutsos, (1985) Access *Methods for Text*", ACM Computing surveys, Vol. 17, No. 1, pp 49 – 74.

[4] Chris Faloutsos, (1985) *Signature files: Design and performance comparison of some signature extraction methods*" Proc. of ACM SIGMOD, pp 63 – 82.

[5] Christos Faloutsos, Stavros Christodoulakis,(1985) *Design of a Signature File Method that Accounts for Non-Uniform Occurrence and Query Frequencies*, Proc. of VLDB, pp 165 – 170.

[6] Christos Faloutsos, Stavros Christodoulakis,(1987) *Description and Performance Analysis of Signature File Methods for Office Filing*, ACM Trans on Office Information Systems, Vol. 5, No. 3, pp 237 – 257.

[7] Christos Faloutsos, Stavros Christodoulakis, (1987) *Optimal Signature Extraction and Information Loss*, ACM Trans. On Database Systems, Vol. 12, No. 3, pp 395 – 428.

[8] Christos Faloutsos, Raphael Chan, (1988) *Fast Text Access Methods for Optical and Large Magnetic Disks: Designs and Performance Comparison*, Proc. of VLDB, pp 280 – 293.

[9] Chris Faloutsos, Raymond Lee, Catherine Plaisant, Ben Shneiderman, (1990)

[10] *Incorporating String Search in a Hypertext System: User Interface and Signature File Design Issues*, Hypermedia, Vol. 2, No. 3, pp 183 – 200.

[11] D. Dervos, Y. Manolopoulous, P. Linardis, (1998) *Comparison of Signature File Methods with Superimposed Coding*, J. Information Processing , 65, pp 101 – 106.

[12] Douglas Comer, (1979) *The Ubiquitos B-Tree*, Computing Surveys, Vol. 11, No. 2, pp121 – 137.

[13] Dik Lun Lee, Young Man Kim, Gaurav Patel, (1995) *Efficient Signature File Methods for Text Retrieval*, IEEE TKDE, Vol. 7, No. 3, pp 423 – 435.

[14] Eleni Tousidou, Alex Nanopoulos, Yannis Manolopoulos, (2000) *Improved Methods for Signature-Tree Construction*, The Computer Journal, Vol. 43, No. 4, pp 301 – 314.

[15] Eleni Tousidou, Panayiotis Bozanis, Yannis Manolopoulos, (2002) *Signature-based structures for objects with set-valued attributes*, Information Systems, Vol. 27, No. 2, pp 93 – 121.

[16] Grady Booch, James Rambaugh, Ivar Jacobson, (2003) *The Unified Modeling Language User Guide*, Pearson Education Pte Ltd.

[17] Hwan-Seung Yong, Sukho Lee, Hyoung-Joo Kim, (1994) *Applying Signatures for Forward Traversal Query Processing in Object-Oriented Databases*, Proc. 10th Intl. conf. Data Engg, pp 518 – 525.

[18] Justin Zobel, Alistair Moffat, Kotagiri Ramamohanarao, (1988) *Inverted Files Versus Signature Files for Text Indexing*, ACM Trans. On Database systems, Vol. 23, No. 4, pp 453 – 490.

[19] A. Kent, R. Sacks-Davis, K. Ramamohanarao, (1990) *A Signature File Scheme Based on Multiple Organizations for Indexing Very Large Text Databases*, J. Am. Soc. Information Science, 1990, Vol. 41, No. 7, pp 508 – 534.

[20] Martin Fowler, Kendall Scott, (2003) *UML Distilled, A brief guide to the standard Object Modeling Language*, II edition, Pearson Education Pte Ltd.

[21] Paolo Ciaccia, Paolo tiberio, Pavel Zezula, (1996) *Declustering of Key-Based Partitioned Signature*

Files, ACM Trans. On Database Systems, Vol. 21, No. 3, pp 295 – 338.

[22] Per-Ake Larson, (1984) *A Method for Speeding up Text Retrieval*, ACM SIGMIS, Database Winter, Vol.15, No. 2, pp 19 – 23.

[23] Rudolf Bayer, Karl Unterauer, (1977) *Prefix B-Trees*, ACM Trans. On Database Systems,Vol. 2, No. 1, pp 11 – 26.

[24] 23. Seyit Kocberber, Fazli Can, (1999) *Compressed Multi-Framed Signature Files: An Index Structure for Fast Information Retrieval*, Proc. ACM Symp. Applied Computing, pp 221 – 226.

[25] 24. Uwe Deppisch, (1986*) S-Tree: A Dynamic Balanced Signature Index for Office Retrieval*, Proc. ACM SIGIR conf, pp 77 – 87.

[26] Walter W. Chang, Hans J. Schek, (1989*) A Signature Access Method for the Starburst Database System*, Proc. of VLDB, 145 – 153.

[27] Wang-chien Lee, Dik L. Lee, (1992) *Signature File Methods for Indexing Object-Oriented Database Systems*, Proc. 2nd Intl. Comp. Sc. Conf, pp 616 – 622.

[28] Yangjun Chen, (2002) *Signature Files and Signature Trees*, Information Processing Letters, 82, pp 213 -221.

[29] Yangjun Chen, Yibin Chen, (2004) *Signature File Hierarchies and Signature Graphs: a New Index Method for Object-Oriented Databases*, Proc. of ACM Symp. on Applied Computing, pp 724 – 728.

[30] Yangjun Chen, (2005) *On the Signature Trees and Balanced Signature Trees*, Proc. of ICDE, pp 742 – 753.

[31] Yangjun Chen, Yibin Chen, (2006) *On the Signature Tree Construction and Analysis*, IEEE TKDE ,Vol.18,No.9, pp 1207 – 1224.

[32] Yoshiharu Ishiwaka, Hiroyuki Kitagawa, Nobuo Ohbo, (1993) *Evaluation of Signature Files as Set Access Facilities in OODBs*, Proc. of ACM SIGMOD, pp 247 – 256.