In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pp. 247–256, Washington, D.C., May 1993.

Evaluation of Signature Files as Set Access Facilities in OODBs

Yoshiharu Ishikawa * Hiroyuki Kitagawa † Nobuo Ohbo †

* Doctoral Degree Program in Engineering, University of Tsukuba, Tsukuba City, Ibaraki 305, Japan
† Institute of Information Sciences and Electronics, University of Tsukuba, Tsukuba City, Ibaraki 305, Japan
e-mail address: {ishikawa, kitagawa, ohbo}@dblab.is.tsukuba.ac.jp

ABSTRACT *Object-oriented database systems (OODBs) need efficient support for manipulation of complex objects. In particular, support of queries involving evaluations of set predicates is often required in handling complex objects. In this paper, we propose a scheme to apply signature file techniques, which were originally invented for text retrieval, to the support of set value accesses, and quantitatively evaluate their potential capabilities. Two signature file organizations, the sequential signature file and the bit-sliced signature file, are considered and their performance is compared with that of the nested index for queries involving the set inclusion operator (⊆). We develop a detailed cost model and present analytical results clarifying their retrieval, storage, and update costs. Our analysis shows that the bit-sliced signature file is a very promising set access facility in OODBs.*

1 INTRODUCTION

Advanced database application areas, such as computer aided design, office automation, and software engineering, handle complex structured data, namely complex objects [Kim88]. The overall goal of object-oriented database system (OODB) researches is to provide efficient complex object management facilities. OODBMSs usually offer data modeling constructs such as the set constructor and the tuple constructor, so that the user can specify structures of complex objects. In order to support complex objects efficiently, it is necessary to investigate methods to accelerate set value manipulation in query processing. As access facilities to complex objects, new indexing mechanisms such as the nested index, the path index and the multiindex have been proposed [Bert89, Stei91]. But these indexes are not designed to fully support set value manipulation in general.

Let us consider a sample database. The schema of the sample database consists of three class definitions:

Student = [*name*: str, *courses*: {**Course**},
hobbies: {*hobby*: str}]
Course = [*name*: str, *category*: str, *teacher*: **Teacher**]
Teacher = [*name*: str, ...]

where "[···]" denotes the tuple constructor, and "{···}" denotes the set constructor. The structure of a class is defined by combining these constructors. For example, **Student** class has a primitive attribute *name* and two set attributes *courses* and *hobbies*. The *courses* attribute takes a set of OIDs of **Course** objects as a value, and the *hobbies* attribute takes a set of strings value. Examples of objects in these classes are shown below:

```
s1:[name:"Jeff", courses:{c1, c3, c4},
    hobbies:{"Baseball", "Fishing"}]
...
c1:[name:"DB Theory", category:"DB", teacher:t1]
...
```

where `s1` denotes an OID of a **Student** object, `c1`, `c3`, and `c4` denote OIDs of **Course** objects and `t1` denotes an OID of a **Teacher** object.

Let us consider the following query for this sample database:

> *Find all students who take all of the lectures in the "DB" category.*

If an access facility which supports set comparison predicates exists on the path "**Student**.*courses*," we can use the following query processing scheme:

1. Retrieve the OIDs of **Course** objects which satisfy the condition '**Course**.*category* = "DB"' into `OID-list`.
2. Retrieve **Student** objects which satisfy the condition '**Student**.*courses* ⊇ `OID-list`.'

Currently proposed indexing techniques such as the nested index are supposed to facilitate the execution of this query. But their usefulness and limitation have not been analyzed well.

Next, let us consider another sample query given as follows:

> *Find all students who take only the lectures in the "DB" category.*

If we replace the search condition '**Student**.*courses* ⊇ `OID-list`' in the above query processing scheme with '**Student**.*courses* ⊆ `OID-list`,' this query can be processed in a similar manner. In general, this query cannot be processed efficiently with the currently proposed indexing techniques.

As exemplified above, facilities to efficiently evaluate set predicates are very important in query processing in OODBs.

In this paper, we propose a scheme to use signature files as set access facilities in OODBs, and quantitatively evaluate their capabilities. The signature file was originally proposed for text data manipulation, and its power has been investigated intensively in the context of word retrieval in the text database [Falo84, Falo87]. However, the signature

file seems also very promising as a set access facility in the context of OODBs. Although the use of signature files for traditional record retrievals has been discussed by some researchers [Pfal80, SD87], study on their capabilities in OODB environments has not been reported to the best of our knowledge. In this paper, we consider the sequential signature file (SSF) and the bit-sliced signature file (BSSF) as two representative signature file organizations. We compare SSF, BSSF, and the nested index (NIX), a well-known indexing method for complex objects, in terms of retrieval cost, storage cost, and update cost. In particular, we analyze retrieval costs for the query involving the set inclusion operator ($\subseteq$) in detail based on the cost model. The analysis clarifies advantages and disadvantages of the three facilities (namely, SSF, BSSF, and NIX) in several situations in query processing in OODBs.

The remainder of this paper is organized as follows. Section 2 introduces sample queries to illustrate two types of query investigated in this paper. Section 3 gives an overview of the signature file techniques and describes their application to set manipulation. The false drop probabilities, which are important performance measures of signature files, are derived for two types of query. Section 4 describes our cost model. Section 5 shows results of intensive analysis of the three set access facilities in terms of retrieval cost, storage cost, and update cost. Section 6 gives a summary and concludes the paper.

2 SAMPLE QUERIES

There exist several set comparison operators used in query evaluation. To examine whether two sets satisfy subset relationship or not, the inclusion operators ($\subset$, $\subseteq$) are used. The membership operator ($\in$) is a special case of the inclusion operator for comparison between a primitive value and a set value. Moreover, the equality operator for the set equality and the overlap operator, which checks whether two sets intersect or not, may appear in queries.

In this paper, we focus on two cases: "$\langle$target set$\rangle \supseteq \langle$query set$\rangle$" and "$\langle$target set$\rangle \subseteq \langle$query set$\rangle$." Two sample queries Q_1 and Q_2 illustrating the two cases for the database introduced in Section 1 are given below (queries are expressed in a SQL-like query language defined in [Kim90]):

Query Q_1 ($T \supseteq Q$): *Find all Students whose hobbies attribute includes* {*"Baseball", "Fishing"*}.

select Student
where hobbies **has-subset** '("Baseball", "Fishing")

We call this type of query "$T \supseteq Q$." T and Q mean "target" and "query," respectively.

Query Q_2 ($T \subseteq Q$): *Find all Students whose hobbies attribute is a subset of* {*"Baseball", "Fishing", "Tennis"*}.

select Student
where hobbies **is-subset**
'("Baseball", "Fishing", "Tennis")

Similarly, we call this type of query "$T \subseteq Q$."

In the following part of this paper, we use Q_1 and Q_2 as typical sample queries involving set predicates, and evaluate the use of signature files in processing such queries.

3 SIGNATURE FILE TECHNIQUES

3.1 Overview and Application to Set Manipulation

The signature file techniques [Falo84, Falo87] are very popular in text retrieval. They require much smaller storage space than inverted files, and can handle insertion and update operations easily. A *signature* is a bit pattern formed for each data object and stored in the *signature file*. In addition to the signature file, there exists an *OID file*, which relates each signature to the OID of a data object. In text retrieval, data objects are usually *logical blocks* consisting of text data including a pre-defined number of words.

A typical query processing with the signature file is as follows: When a query is given, a *query signature* is formed from the query value. Then each signature in the signature file is examined over the query signature. If the signature satisfies a pre-defined condition, the corresponding data object becomes a candidate that may satisfy the query. Such a data object is called a *drop*. The next step of query processing is the *false drop resolution*. Each drop is accessed and examined whether it actually satisfies the query condition. Drops that fail the test are called *false drops*, while the qualified data objects are called *actual drops*.

The *superimposed coding method* is often used to form a signature [Falo85]. In text retrieval, a logical block consists of textual data including a pre-defined constant number of words. In case of text retrieval, a *word signature* is first created for each word in a logical block. Word signatures have F bits length and contain m "1"'s and $F - m$ "0"'s. Then, a *block signature* is composed by OR-ing word signatures for each logical block and stored in the signature file. F and m are design parameters tuned based on performance perspective.

We utilize the superimposed coding-based signature file technique to support processing queries including set predicates such as Q_1 and Q_2. In cases of Q_1 and Q_2, we create a signature file which will facilitate searching qualified *hobbies* set attribute values. We call *hobbies* an *indexed set attribute*. The superimposed coding method is used to form a *set signature* for each indexed set attribute value. Each element in a set value yields an *element signature* of size F, with m bits set to "1." Then, element signatures are OR-ed together to form a set signature. The set signatures made from indexed set attribute values are called *target signatures* and are stored in the signature file.

When a query is given, a *query signature* is formed from the query value and then the signature file is examined. If a target signature satisfies the condition implied by the set predicate in the query, the corresponding data object, which is a **Student** object in our case, becomes a candidate which may satisfy the query. The signature file search condition for the query type $T \supseteq Q$ is given as follows:

> *For all bit positions which are set to "1" in the query signature, if "1"'s are set in a target signature, the corresponding data object becomes a drop.*

Figure 1 illustrates how actual drops and false drops occur for the query Q_1. For the two elements in the query set {"Baseball", "Fishing"}, element signatures `"01000100"` and `"00010100"` are created, respectively. Then, a query signature `"01010100"` is composed by bitwise-OR-ing the element signatures. Now suppose that a target signature `"01110101"`, composed from a target set {"Baseball", "Golf", "Fishing"}, is stored in the signature file. Then, the target signature satisfies the above search condition and "$\langle$target set$\rangle \supseteq \langle$query set$\rangle$" actually holds. Therefore, this causes an actual drop. On the other hand, let us consider a target signature `"11010101"`, composed from a target set {"Baseball", "Football", "Tennis"}. Although this target set does not satisfy the query condition, the target signature satisfies the above search condition. Therefore, a false drop

occurs in this case. This is due to the hash collisions and the superimposed coding method.

Query Element	Signature
Baseball	01000100
Fishing	00010100
Query Signature →	01010100

Actual Drop		**False Drop**	
Target Element	Signature	Target Element	Signature
Baseball	01000100	Baseball	01000100
Golf	00100001	Football	10000100
Fishing	00010100	Tennis	00010001
Target Signature →	01110101	Target Signature →	11010101

Figure 1: Actual Drop and False Drop ($T \supseteq Q$)

Next, let us consider that the query Q_2 ($T \subseteq Q$). In this case, the search condition for the query is given as follows:

> *For all bit positions which are set to "1" in a target signature, if "1"'s are set in the query signature, the corresponding data object becomes a drop.*

Figure 2 illustrates how actual drops and false drops occur for this case. A query signature "11010101" is composed from the given query set {"Baseball", "Football", "Tennis"}. Let us consider a target signature "11000100", composed from a target set {"Baseball", "Football"}. Then, the target signature satisfies the above search condition and "⟨target set⟩ ⊆ ⟨query set⟩" actually holds. Therefore, this causes an actual drop. On the other hand, the target signature "01010100" also satisfies above condition, although the corresponding target set {"Baseball", "Fishing"} does not satisfy the query condition. Therefore, this causes a false drop.

Query Element	Signature
Baseball	01000100
Football	10000100
Tennis	00010001
Query Signature →	11010101

Actual Drop		**False Drop**	
Target Element	Signature	Target Element	Signature
Baseball	01000100	Baseball	01000100
Football	10000100	Fishing	00010100
Target Signature →	11000100	Target Signature →	01010100

Figure 2: Actual Drop and False Drop ($T \subseteq Q$)

There are a number of choices in physical signature file organizations. In this paper, we consider two well-known storage organizations for signature files: *Sequential Signature File* (SSF) and *Bit-Sliced Signature File* (BSSF). Figure 3 illustrates the file structures of SSF and BSSF. SSF is the simplest organization, which is easy to implement and requires low storage space and low update cost. Set signatures are stored sequentially in the signature file. When a query is given, a full scan of the signature file is required. Therefore, it is generally slow in retrieval.

BSSF stores signatures in a column-wise manner. Thus, F files (called *bit-slice files*), one per each bit position of the set signatures, are used. In retrieval, only a part of the F bit-slice files have to be scanned, so that the search cost is lower than that of SSF. However, update cost becomes larger. For example, an insertion of a new set signature requires about F disk accesses, one for each bit-slice file.

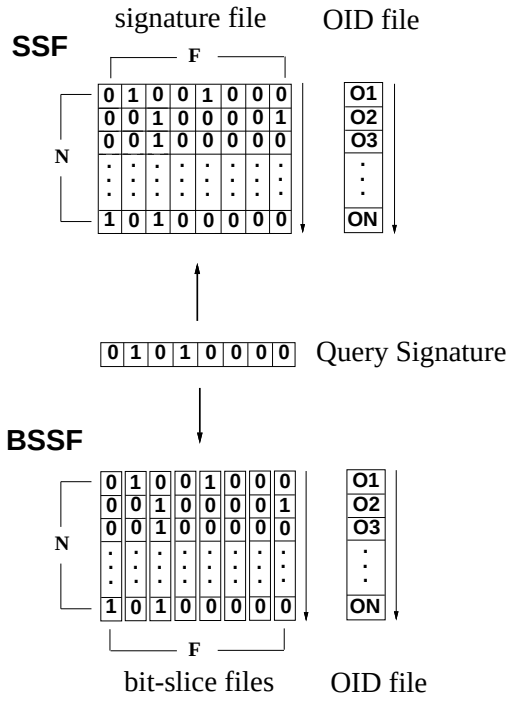

Figure 3: SSF and BSSF

3.2 False Drop Probability

In the following two subsections, we estimate the false drop probabilities for queries $T \supseteq Q$ and $T \subseteq Q$. False drop probability is an important measure to estimate the performance of signature files. Formally, the *false drop probability* F_d is given by the following expression [Falo88]:

$$F_d = \frac{\text{false drops}}{N - \text{actual drops}},$$

where N is the number of data objects.

Table 1 gives symbols used in the estimations.

Symbol	Definition
F_d	false drop probability
F	signature size in bits
m	number of "1"'s in an element signature
m_t	expected number of "1"'s in a target signature
m_q	expected number of "1"'s in a query signature
D_t	cardinality of a target set
D_q	cardinality of a query set
b_t^i	ith bit position in a target signature
b_q^i	ith bit position in a query signature

Table 1: Symbols

3.2.1 False Drop Probability for $T \supseteq Q$

First, we estimate the false drop probability F_d for the query $T \supseteq Q$ (Q_1). As discussed in [Falo84], we can derive F_d assuming the case of unsuccessful search, where we have no actual drops. In Section 4, the retrieval cost for the query $T \supseteq Q$ is derived based on the false drop probability given here.

In the case of unsuccessful search, as shown in Figure 1, a false drop occurs when the following condition holds for every bit position i ($1 \le i \le F$):

$$b_q^i = 1 \quad \Rightarrow \quad b_t^i = 1.$$

This condition is equivalent to

$$b_t^i = 0 \quad \Rightarrow \quad b_q^i = 0. \tag{1}$$

We derive the false drop probability F_d based on the latter condition.

In the following analysis, we assume that the "1"'s are uniformly distributed in an element signature. In other words, we suppose that the hash function, which is used to form element signatures, has ideal characteristics. Therefore, the probability that a bit position b in an element signature is set to "1" is given by m/F. Let m_t the expected number of bits in a target signature which are set to "1" and let D_t the cardinality of the target set. Then, we have

$$\begin{aligned} m_t &= F\mathrm{Prob}\{b_t^i = 1\} \\ &= F\left[1 - (1 - \tfrac{m}{F})^{D_t}\right]. \end{aligned}$$

Therefore, the expected number of bits which are set to "0" is given by

$$\begin{aligned} F - m_t &= F\left(1 - \tfrac{m}{F}\right)^{D_t} \\ &\approx Fe^{-\frac{m}{F}D_t} \quad \left(\tfrac{m}{F} \ll 1\ ^{(\ddagger)}\right). \end{aligned}$$

Since a false drop occurs on the above condition (1), the false drop probability F_d is given as follows:

$$\begin{aligned} F_d &= \mathrm{Prob}\{b_q^1 = 0 \wedge \cdots \wedge b_q^{F-m_t} = 0\} \\ &= \mathrm{Prob}\{b_q^1 = 0 \wedge \cdots \wedge b_q^{Fe^{-\frac{m}{F}D_t}} = 0\}. \end{aligned}$$

If $\frac{m}{F-k+1} \ll 1$ holds for $1 \le k \le Fe^{-\frac{m}{F}D_t}$ $^{(\ddagger)}$, we can use the approximate expression (9) (see APPENDIX A). Therefore, we have

$$F_d \approx (1 - e^{-\frac{m}{F}D_t})^{mD_q}. \tag{2}$$

When we consider F_d given by equation (2) as a function of m, F_d takes the minimum value for the m value given below. Let us consider the following function f with respect to m:

$$\begin{aligned} f(m) &= \ln(F_d) \\ &= D_q \ln(1 - e^{-\frac{m}{F}D_t})^m, \end{aligned}$$

where ln stands for the natural logarithm. Since the expression $(1 - e^{-\frac{m}{F}D_t})^m$ takes the minimum value when m is

$$m_{opt} = \frac{F \ln 2}{D_t} \tag{3}$$

[Falo84], F_d also becomes the minimum value when $m = m_{opt}$. Namely, if F and D_t are given, the value of m which minimizes the false drop probability is given by equation (3). In this case, we can simplify equation (2) as follows.

$$F_d \approx \left(\tfrac{1}{2}\right)^{\frac{D_q}{D_t} F \ln 2} \tag{4}$$

In the use of signature files in the text database, the m value is often determined by equation (3).

$^{(\ddagger)}$In the following cost analysis, these conditions are satisfied.

3.2.2 False Drop Probability for $T \subseteq Q$

Now we consider the false drop probability F_d for the query $T \subseteq Q$ (Q_2) in a similar way.

A false drop for $T \subseteq Q$ occurs when the following condition holds for every bit position i ($1 \le i \le F$):

$$b_t^i = 1 \quad \Rightarrow \quad b_q^i = 1.$$

This condition is equivalent to

$$b_q^i = 0 \quad \Rightarrow \quad b_t^i = 0. \tag{5}$$

In a similar way to $T \supseteq Q$, we can derive the false drop probability F_d for $T \subseteq Q$ based on condition (5). If $\frac{m}{F-k+1} \ll 1$ holds for $1 \le k \le Fe^{-\frac{m}{F}D_q}$ $^{(\ddagger)}$, the false drop probability F_d is given by:

$$\begin{aligned} F_d &= \mathrm{Prob}\{b_t^1 = 0 \wedge \cdots \wedge b_t^{F-m_q} = 0\} \\ &\approx (1 - e^{-\frac{m}{F}D_q})^{mD_t}. \end{aligned} \tag{6}$$

We can derive the m value which minimizes equation (6) as follows:

$$m_{opt} = \frac{F \ln 2}{D_q}$$

However, it is hard to use the above value in the design of signature files, since D_q usually varies depending on queries.

4 COST MODEL

In this section, we describe our cost model to compare the sequential signature file (SSF), the bit-sliced signature file (BSSF), and the nested index (NIX) as set access facilities for queries $T \supseteq Q$ and $T \subseteq Q$. Table 2 lists values for the constant parameters used in the following analysis, and Table 3 shows symbols used to measure the performance.

Symbol	Definition and Value
N	total number of objects (= 32,000)
P	size of a disk page in bytes (= 4096)
oid	size of an OID in bytes (= 8)
V	cardinality of the set domain (= 13,000)
b	number of bits per byte (= 8)
O_n	number of OIDs in a disk page (= $\lfloor P/oid \rfloor$ = 512)
SC_{OID}	size of an OID file in pages (= $\lceil N/O_n \rceil$ = 63)
P_u	number of page accesses per object on unsuccessful retrieval (= 1)
P_s	number of page accesses per object on successful retrieval (= 1)

Table 2: Constant Parameters

Symbol	Definition
RC	total retrieval cost
SC	total storage cost
A	total number of actual drops
α	number of actual drops per page of the OID file
LC_{OID}	look-up cost for the OID file
SC_{SIG}	storage cost for signature file
UC_I	insert cost of one set value
UC_D	deletion cost of one set value

Table 3: Performance Measures

Here, we make some assumptions:

- The database contains N objects. Each object has an indexed set attribute whose value is a set of the same cardinality D_t. Namely, D_t elements are contained in each set attribute value. In the following analysis, we consider two cases of $D_t = 10$ and $D_t = 100$.

- The elements of a set attribute value are randomly selected from a domain whose cardinality is V.
- Objects are straightforwardly stored in the object file; no type of decomposition is applied.
- Each object has an unique OID. We can directly access any object by its OID.

In the following three subsections, we estimate the retrieval cost, the storage cost, and the update cost for each access facility. Since the performance of these access facilities mainly depends on the I/O cost, we estimate the costs in terms of the number of page accesses.

We classify updates into two cases: "insertion of a new set value," and "deletion of a set value." UC_I and UC_D denote the costs for insertion and deletion, respectively.

4.1 Cost Estimation of SSF

SSF consists of two files: a signature file and an OID file as illustrated in Figure 3. Set signatures and OIDs are sequentially stored in each file.

Retrieval Cost: Object retrieval using SSF is processed as follows:

1. A query signature is formed from a given query set value.
2. SSF is scanned sequentially to examine each target signature. If the target signature satisfies the query condition as explained in Subsection 3.1, the corresponding OID is retrieved from the OID file into `OID-list`.
3. For each OID entry in `OID-list`, the object is retrieved and checked whether it actually satisfies the query condition. Qualified objects are returned to the user.

We derive the retrieval cost based on the discussion in [Falo88] as follows:

$$RC = SC_{SIG} + LC_{OID} + P_s A + P_u F_d (N - A), \quad (7)$$

where SC_{SIG} is the storage cost of the signature file. Since SSF requires a full scan over the signature file, SC_{SIG} directly influences the total retrieval cost RC. LC_{OID} is the look-up cost for the OID file and its value is given below. $P_s A + P_u F_d (N - A)$ is cost for the false drop resolution step. $P_s A$ is the object retrieval cost for actual drops, since A objects (actual drops) have to be retrieved. Similarly, $P_u F_d (N - A)$ is the object retrieval cost for false drops. $F_d (N - A)$ is the number of the false drops.

The look-up cost of the OID file is

$$LC_{OID} = SC_{OID} \times min(F_d (O_n - \alpha) + \alpha, 1),$$

where α is the expected number of actual drops per page of the OID file, and given as follows:

$$\alpha = \frac{A}{SC_{OID}}.$$

For each page of the OID file, we have α OIDs of actual drops and $F_d (O_n - \alpha)$ OIDs of false drops. When $F_d (O_n - \alpha) + \alpha \geq 1$, we can access all such OIDs by one page access, so that the look-up cost per OID file page is at most 1 (page).

Storage Cost: The storage cost of SSF is given as follows:

$$\begin{aligned} SC &= SC_{SIG} + SC_{OID} \\ SC_{SIG} &= \left\lceil \frac{NF}{Pb} \right\rceil, \end{aligned}$$

where SC_{SIG} is the storage cost of the signature file, and SC_{OID} is the storage cost of the OID file.

Update Cost: The update cost of SSF is as follows:

$$\begin{aligned} UC_I &= 2 \\ UC_D &= \frac{SC_{OID}}{2}. \end{aligned}$$

Insertion of a set value requires one page access both to the signature file and to the OID file to append the information at the end of the files. In deleting a set value, a delete flag is set in the OID file. Therefore, $\frac{SC_{OID}}{2}$ page accesses are needed.

4.2 Cost Estimation of BSSF

BSSF consists of F different bit-slice files and an OID file as illustrated in Figure 3.

Retrieval Cost: Object retrieval using BSSF slightly differs for $T \supseteq Q$ and $T \subseteq Q$, and is specified as follows:

1. A query signature is formed from a given query set value.
2. If the query is $T \supseteq Q$:
   (a) For each bit position which is set to "1" in the query signature, the bit-slice is retrieved from the corresponding bit-slice file. The expected number of bit-slice files to be accessed is m_q given below.
   (b) These bit-slices are AND-ed together. For each entry where "1" is set in the resulting AND-ed bit-slice, retrieve the corresponding OID from the OID file into `OID-list`.
3. If the query is $T \subseteq Q$:
   (a) For each bit position which is set to "0" in the query signature, the bit-slice is retrieved from the corresponding bit-slice file. The expected number of bit-slice files to be accessed is $F - m_q$.
   (b) These bit-slices are OR-ed together. For each entry where "0" is set in the resulting OR-ed bit-slice, retrieve the corresponding OID from the OID file into `OID-list`.
4. For each OID entry in `OID-list`, the object is retrieved and checked whether it actually satisfies the query condition. Qualified objects are returned to the user.

The retrieval cost of BSSF also differs for $T \supseteq Q$ and $T \subseteq Q$.

$T \supseteq Q$:

$$RC = \left\lceil \frac{N}{Pb} \right\rceil m_q + LC_{OID} + P_s A + P_u F_d (N - A)$$

$T \subseteq Q$:

$$\begin{aligned} RC = \left\lceil \frac{N}{Pb} \right\rceil (F - m_q) + LC_{OID} + P_s A \\ + P_u F_d (N - A) \end{aligned} \quad (8)$$

Here, m_q is derived similarly to m_t (see Subsection 3.2) and given by the following equation:

$$\begin{aligned} m_q &= F(1 - (1 - \tfrac{m}{F})^{D_q}) \\ &\approx F(1 - e^{-\frac{m}{F} D_q}) \qquad \left(\tfrac{m}{F} \ll 1\right) \end{aligned}$$

$\left\lceil \frac{N}{Pb} \right\rceil$ is the storage cost per bit-slice file and becomes 1 for the parameter values in Table 2.

Storage Cost: The storage cost of BSSF is given as follows:

$$SC = \left\lceil \frac{N}{Pb} \right\rceil F + SC_{OID}$$

Update Cost: The update cost of BSSF is given below:

$$UC_I = F + 1$$
$$UC_D = \frac{SC_{OID}}{2}.$$

To insert a set value, insertions to all the F bit-slice files and one page access to the OID file are needed. To delete a set value, $\frac{SC_{OID}}{2}$ page accesses are required to set a delete flag in the OID file.

4.3 Cost Estimation of NIX

NIX is an index mechanism based on the B-tree. In a leaf page, it stores index entries composed of a key value and the list of OIDs for objects which have the key value in the indexed nested attribute. The format of a nonleaf node is similar to that of a B-tree.

Reconsider the sample database. Suppose that an NIX is created on the path "**Student**.*courses.category*." Leaf pages of this NIX contain entries such as `["DB", {s1, s2}]`, where `s1` and `s2` are the OIDs of **Student** objects. To support the sample queries Q_1 and Q_2, we use an NIX on the path "**Student**.*hobbies.hobby*." Leaf pages of this NIX will contain entries such as `["Baseball", {s1, s2}]`.

[Bert89] compared NIX with the path index and the multiindex in the context of navigation queries in OODBs. In the following, we develop cost estimation of NIX for the queries $T \supseteq Q$ and $T \subseteq Q$ extending the cost model introduced in [Bert89]. Parameter symbols for NIX are listed in Table 4. Other parameters are similar to the case of signature files.

Symbol	Definition and Value
ll	size of a leaf node index entry for NIX
d	average number of objects whose indexed set attribute includes a given set element value
kl	size of a key value for NIX (= 8 bytes)
$noid$	size of a field which specifies the number of OID entries (= 2 bytes)
lp	number of leaf level index pages
nlp	number of nonleaf level index pages
f	average fanout from a nonleaf node (= 218)
rc	retrieval cost per index entry

Table 4: Parameters for NIX

The size of a leaf entry is

$$ll = d \times oid + kl + noid,$$

where d denotes the average number of objects whose indexed set attribute value includes a given set element value, and is assumed as follows:

$$d = \frac{D_t N}{V}.$$

Therefore, ll is

$$ll = \frac{D_t N}{V} \times oid + kl + noid.$$

Now we assume that for each value in the set domain, at least one corresponding OID exists. Then, the total number of leaf pages is

$$lp = \left\lceil \frac{V}{\lfloor \frac{P}{ll} \rfloor} \right\rceil.$$

The number of nonleaf pages is given as follows:

$$nlp = \left\lceil \frac{lp}{f} \right\rceil + \left\lceil \frac{\lceil lp/f \rceil}{f} \right\rceil + \cdots + X,$$

where f is a fanout of NIX and X is a integer value which satisfies $1 \leq X < f$. If X is not 1, 1 is added to nlp for the root node page.

Retrieval Cost: Object retrieval using NIX differs for $T \supseteq Q$ and $T \subseteq Q$, and is specified as follows:

$T \supseteq Q$:

1. For each element in the query set, retrieve the OIDs corresponding to the element using NIX.
2. Take the intersection of the OID sets. For each OID in the resulting set, retrieve the object and return them to the user.

$T \subseteq Q$:

1. For each element in the query set, retrieve the OIDs corresponding to the element using NIX.
2. Take the union of the OID sets. For each OID in the resulting set, retrieve the object and check whether it actually satisfies the query condition. Qualified objects are returned to the user.

For $D_t = 10$ and $D_t = 100$, the height of an NIX becomes 2. Therefore, the NIX look-up cost for a set element is $rc = 2 + 1 = 3$ (pages). Thus, the retrieval cost of NIX is given as follows:

$T \supseteq Q$:

$$RC = rcD_q + P_s A$$

$T \subseteq Q$:

$$RC = rcD_q + P_u N \frac{\sum_{i=1}^{D_t - 1} ({}_{D_q}C_i \times {}_{V-D_q}C_{D_t - i})}{{}_V C_{D_t}} + P_s N \frac{{}_{D_q}C_{D_t}}{{}_V C_{D_t}}$$

The derivation of this equation is shown in APPENDIX B.

Storage Cost: The storage cost of NIX is $SC = lp + nlp$. For $D_t = 10$ and $D_t = 100$, the storage costs are shown in Table 5.

D_t	lp	nlp	SC
10	685	5	690
100	6500	31	6531

Table 5: Storage Cost of NIX

Update Cost: The update costs of NIX are

$$U_I = rcD_t$$
$$U_D = rcD_t,$$

on condition that we do not consider node splits. Insertion and deletion of a set value require D_t insertions/deletions.

4.4 Actual Drop

False drop probabilities are formulated in Subsection 3.2. False drops occur only in signature files. However, actual drops are related to all three access facilities investigated here. In this subsection, we estimate the actual drop A for $T \supseteq Q$ and $T \subseteq Q$.

$T \supseteq Q$: In this case, we assume that $D_t \geq D_q$. Since the cardinality of the set domain is V, the probability that a query set value becomes a subset of a target set is

$$\frac{{}_{V-D_q}C_{D_t - D_q}}{{}_V C_{D_t}}.$$

The actual drop A is the expected number of such target sets. Therefore, we have

$$A = N \frac{{}_{V-D_q}C_{D_t - D_q}}{{}_V C_{D_t}}.$$

$T \subseteq Q$: We assume that $D_q \geq D_t$. The probability that a query set value becomes a superset of a target set is

$$\frac{{}_{D_q}C_{D_t}}{{}_{V}C_{D_t}}.$$

Therefore, we have

$$A = N \frac{{}_{D_q}C_{D_t}}{{}_{V}C_{D_t}}.$$

This actual drop value is almost negligible for probable values of D_t and D_q.

5 COST ANALYSIS

In this section, we compare the retrieval costs of SSF, BSSF, and NIX for queries $T \supseteq Q$ and $T \subseteq Q$. We consider two cases of $D_t = 10$ and $D_t = 100$. Due to the space limitation, we present only an essential part of the analysis results.

5.1 Retrieval Cost for $T \supseteq Q$

5.1.1 Overall Trend of Retrieval Cost

First, we analyze the retrieval cost RC of SSF, BSSF, and NIX for $T \supseteq Q$. Figure 4 shows the retrieval costs of three set access facilities under $D_t = 10$. D_q varies from 1 to 10. For signature files, there exist two design parameters F and m, so that we can tune the performance. In this case, we have used two signature sizes $F = 250$ (bits) and $F = 500$ (bits), and m_{opt}, given by equation (3), as the value of m. In the context of text retrieval, m_{opt} is a typical m value. When $m = m_{opt}$, the false drop probability F_d given by equation (4) is almost negligible for the parameters in this analysis.

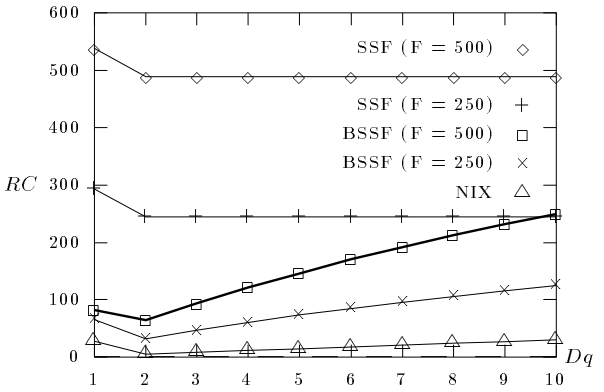

Figure 4: Retrieval Cost RC ($D_t = 10$)

In the case of Figure 4, the retrieval costs of SSF and BSSF are higher than that of NIX. Since SSF requires a full scan over the signature file in retrieval, SC_{SIG}, the storage cost of the signature file, directly influences the retrieval cost. If we choose a smaller signature size F, the storage cost might decrease. However, the false drop probability will increase. This is a dilemma of SSF. On the other hand, the retrieval cost of BSSF does not depend on the storage cost but on m_q, the expected number of "1"'s in the query signature. As D_q increases, m_q increases. Therefore, the retrieval cost of BSSF becomes higher for larger D_q as shown in the Figure 4. However, we can improve the retrieval cost of BSSF as discussed in Subsection 5.1.2.

Similar properties of the three set access facilities are observed for $D_t = 100$. We leave out details of analysis for the case of $D_t = 100$.

5.1.2 Tuning m-value for BSSF

The retrieval cost of BSSF is directly influenced by the m value. Although m_{opt} gives the minimum false drop probability, it is not necessarily an optimal choice for the total retrieval cost. If we take a smaller m value than m_{opt}, we first get a lower total retrieval cost, even if the false drop probability becomes worse. However, if m becomes too small, the total cost increases drastically because the false drop probability becomes intolerable.

For $D_t = 10$ and $D_t = 100$, we have compared the retrieval cost of BSSF with a small m value with that of NIX. Figure 5 shows the retrieval cost for $D_t = 10$ and $F = 500$. In this case, we use the equation (2) as the false drop probability and the value of m ranges from 1 to 4. As shown in the figure, when $D_q = 1$, BSSF is inferior to NIX. However, for other values of D_q, the retrieval cost of BSSF with a small m value is comparable to or lower than that of NIX. In other cases (e.g. $D_t = 100$, $F = 2500$), we have obtained similar results showing an advantage of BSSF with a small m value.

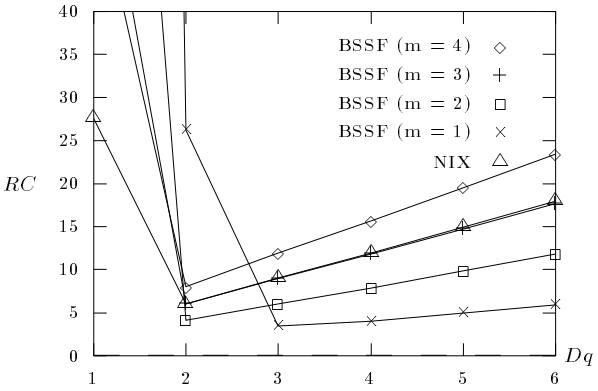

Figure 5: Retrieval Cost RC ($D_t = 10$, $F = 500$)

5.1.3 Smart Object Retrieval

If we observe Figure 5 carefully, we can see that there exist efficient object retrieval strategies for BSSF and NIX which reduce the expected number of page accesses.

Suppose that we have a BSSF with $m = 2$ and a query with $D_q = 3$ is issued by the user. If the object retrieval is done as described in Subsection 4.2, it costs 6.0 page accesses as shown in Figure 5. However, note that we do not always have to look up all the m_q bit-slice files to answer the query. We may use only part of the m_q bit-slice files to select candidate objects, since the final qualification of the candidates is done at the false drop resolution step in any case. In this example, if only two elements in the query set are used to form a query signature, the total cost decreases to 4.0 pages, as indicated by the page accesses for $D_q = 2$ and $m = 2$. Although the increase of D_q leads the reduction of false drops and actual drops, it also brings about the increase of m_q. Therefore, it is not wise to stick to looking up all the m_q bit-slice files, as exemplified by the observation here.

The above discussion suggests the following smart object retrieval strategy for BSSF ($m = 2$):

1. If $D_q = 1$ or $D_q = 2$, look up BSSF as described in Subsection 4.2.
2. If $D_q \geq 3$, form a query signature from only two arbitrary elements in the query set value and look up BSSF using this query signature.

Under this smart strategy, the retrieval cost become constant for $D_q \geq 2$.

Figure 5 indicates that we can also devise a smart object retrieval strategy for NIX in a similar way. The smart strategy uses NIX as follows:

1. If $D_q = 1$ or $D_q = 2$, use NIX as described in Subsection 4.3. D_q index look-ups are needed.
2. If $D_q \geq 3$, look up NIX only two times for two arbitrary elements in the query set value. Then, take the intersection of the two OID sets. Finally, for each OID in the resulting OID set, retrieve the object and check whether it actually satisfies the query condition.

The retrieval costs under these smart object retrieval strategies are shown in Figure 6 ($D_t = 10$) and Figure 7 ($D_t = 100$). As shown in these figures, NIX has an advantage only for $D_q = 1$. BSSF shows almost equal or lower retrieval cost for $D_q \geq 2$ in Figure 6 and $D_q \geq 3$ in Figure 7. Therefore, BSSF is comparable to NIX for the query $T \supseteq Q$.

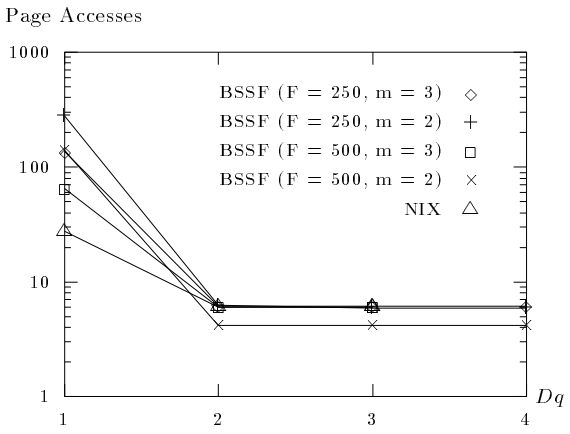

Figure 6: Smart Retrieval Cost ($D_t = 10$)

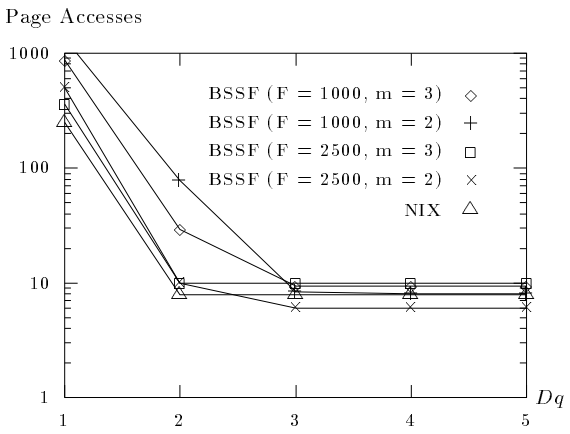

Figure 7: Smart Retrieval Cost ($D_t = 100$)

5.2 Retrieval Cost for $T \subseteq Q$

5.2.1 Overall Trend of Retrieval Cost

Now we analyze the retrieval cost RC for $T \subseteq Q$ for two D_t values, $D_t = 10$ and $D_t = 100$. First, let us observe the overall trend of the retrieval costs of SSF, BSSF and NIX.

Figure 8 shows the retrieval costs of the three access facilities for $D_t = 10$ and $F = 500$. D_q varies from 10 ($= D_t$) to 1000.

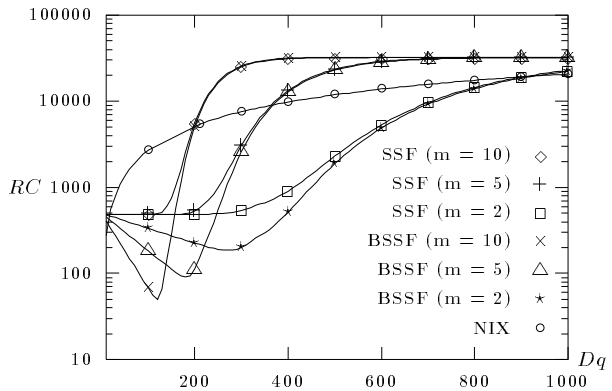

Figure 8: Retrieval Cost RC ($D_t = 10$)

First, let us compare SSF and BSSF. For large D_q values, the retrieval costs of SSF and BSSF are high and monotonically increase as D_q becomes larger. Both the retrieval costs of SSF and BSSF, given by equations (7) and (8), come close to $P_u N$ for large D_q values. The reason is that the false drop probability is almost 1 for large D_q (Recall that actual drop A is negligible in this case). In these situations, most of the data objects have to be accessed to answer the query. For small D_q values, the retrieval cost of BSSF is superior to that of SSF. This is due to the difference of the file organizations; SSF requires a full scan over the signature file, whereas BSSF only requires accesses to $F - m_q$ bit-slice files. For all D_q values, Figure 8 shows superiority of BSSF over the corresponding SSF.

The retrieval cost of NIX monotonically increases as D_q increases. As described in Subsection 4.3, D_q look-ups of NIX is performed for the query $T \subseteq Q$, and then the union of the retrieved OID sets is taken. As D_q increases, the cardinality of the resulting OID set comes close to N.

For other values of F and D_t, similar results are observed, and the explanation is omitted here.

5.2.2 Smart Object Retrieval

As in case of $T \supseteq Q$, we can also improve the object retrieval strategy of BSSF for the query $T \subseteq Q$. In the following, we describe the smart object retrieval strategy and compare the retrieval cost of BSSF (with the smart strategy) with that of NIX.

Note that the graph of BSSF with $m = 2$ in Figure 8 has the minimum value for $D_q \simeq 300$. When we compare the object retrieval costs for $D_q = 100$ and $D_q = 300$, the difference is mainly due to the number of bit-slice pages to be retrieved (i. e. $\lceil \frac{N}{Pb} \rceil (F - m_q)$); it costs 335 pages for $D_q = 100$, and 150 pages for $D_q = 300$. Note that, for $D_q \leq 300$, the numbers of the false drops and the actual drops are almost 0. Therefore, $335 - 150 = 185$ (pages) are the page accesses for the useless bit-slices. Namely, even if we look up such bit-slices to lessen the number of drops, it does not pay because the number of drops is already almost 0.

Based on the above observation, a smart object retrieval strategy for $T \subseteq Q$ is given as follows:

1. If $D_q \leq D_q^{opt}$, look up only $Fe^{-\frac{m}{F}D_q^{opt}}$ bit-slices out of $F - m_q$ bit-slices which correspond to the bit positions set to "0" in the query signature. These $Fe^{-\frac{m}{F}D_q^{opt}}$ bit-slices can be selected arbitrarily. The remaining object access process is similar to that described in Subsection

4.2.

2. If $D_q > D_q^{opt}$, retrieve objects as described in Subsection 4.2.

Here, D_q^{opt} is the value of D_q which minimizes the total retrieval cost of BSSF (see APPENDIX C).

The retrieval costs of BSSF under the smart object retrieval strategy are shown in Figure 9 ($D_t = 10$) and Figure 10 ($D_t = 100$). We have used small m values for consistency with the analysis in Subsection 5.1. For comparison, the retrieval cost of NIX is also plotted. The retrieval cost becomes constant for $D_q \leq D_q^{opt}$. This is the effect of the smart object retrieval strategy. Since D_q^{opt} is relatively larger than D_t, the retrieval cost of BSSF is considered to be a constant value for probable values of D_q. On the other hand, the retrieval cost of NIX is large even with the smaller values of D_q. The analyses show that BSSF is the most efficient set access facility concerning the query $T \subseteq Q$.

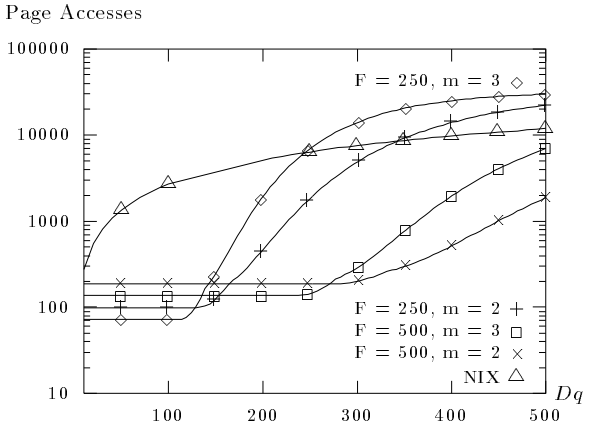

Figure 9: Smart Retrieval Cost ($D_t = 10$)

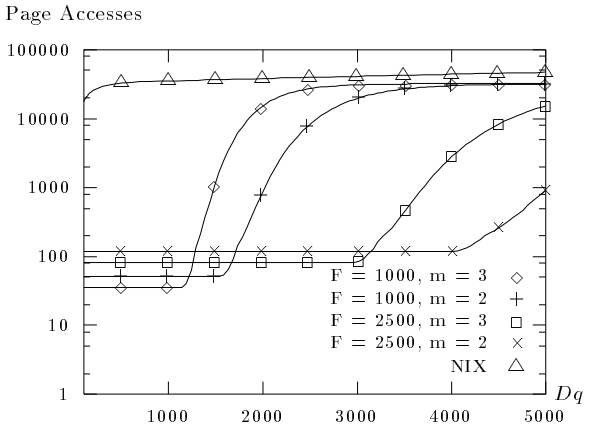

Figure 10: Smart Retrieval Cost ($D_t = 100$)

5.3 Storage Cost

Table 6 shows the storage costs of the three set access facilities for several parameter values. The storage costs have been calculated from the equations in Section 4.

5.4 Update Cost

Table 7 shows the update costs UC_I and UC_D for the three access facilities.

$D_t = 10$

File	SC
SSF ($F = 250$)	307
SSF ($F = 500$)	551
BSSF ($F = 250$)	313
BSSF ($F = 500$)	563
NIX	690

$D_t = 100$

File	SC
SSF ($F = 1000$)	1040
SSF ($F = 2500$)	2504
SSF ($F = 5000$)	4946
BSSF ($F = 1000$)	1063
BSSF ($F = 2500$)	2563
NIX	6531

Table 6: Storage Cost

File	UC_I	UC_D
SSF	2	31.5
BSSF ($F = 250$)	251	31.5
BSSF ($F = 500$)	501	31.5
BSSF ($F = 1000$)	1001	31.5
BSSF ($F = 2500$)	2501	31.5
NIX ($D_t = 10$)	30	30
NIX ($D_t = 100$)	300	300

Table 7: Update Cost

6 SUMMARY AND CONCLUSION

In this paper, we have applied the signature file techniques to support processing queries including set predicates in OODBs. Two signature file organizations, the sequential signature file (SSF) and the bit-sliced signature file (BSSF) were compared with the nested index (NIX). We have developed a cost model and analyzed retrieval, storage, and update costs of the three set access facilities for two types of queries, $T \supseteq Q$ and $T \subseteq Q$.

Our cost model has shown that the storage costs of SSF, BSSF, and NIX become higher in this order. Although SSF and BSSF are, in general, almost comparable, the storage cost of NIX is much higher than that of BSSF. For $D_t = 10$, we have used two F values, $F = 250$ and $F = 500$. The storage costs of BSSF are about 45% (for $F = 250$) and 80% (for $F = 500$) of that of NIX. For $D_t = 100$, we have used two F values, $F = 1000$ and $F = 2500$. Their storage costs are about 16% and 38% of that of NIX, respectively.

The update cost of SSF is relatively low in comparison with those of BSSF and NIX. The deletion costs of BSSF depends only on the size of the OID file and is equal to that of SSF. It does not depend on F values. However, the insertion cost of BSSF is almost proportional to F, since it requires about F page accesses for insertion. Note that these insert costs of BSSF are based on the worst case assumption. Therefore, it may be possible to improve the insertion cost. The insertion cost and the deletion cost of NIX are equal, and proportional to D_t.

As for the retrieval cost, the analysis shows that SSF is inferior to BSSF both for the queries $T \supseteq Q$ and $T \subseteq Q$. SSF requires a full scan over the signature file, whereas only a part of the bit-slice files are accessed in BSSF. Moreover, there exist smart object retrieval strategies which improve the retrieval cost of BSSF as discussed in Section 5. Therefore, as far as the retrieval cost is concerned, BSSF is more appropriate as a set access facility.

For the query $T \supseteq Q$, the retrieval cost of BSSF with a small m value is almost equal to that of NIX except for $D_q = 1$. Under the smart object retrieval strategies, their retrieval costs are constants for most of the D_q values, and the constant cost values are almost the same in cases discussed in Section 5. However, for very small D_q values, the retrieval cost of BSSF becomes worse due to the false drops. In particular, for $D_q = 1$, NIX is more efficient than BSSF in all cases investigated. For the query $T \subseteq Q$, BSSF costs a small constant amount of page accesses for probable

values of D_q, and overwhelms NIX.

From these analyses, we can conclude that BSSF with a small m is a very promising set access facility in OODBs. Let us focus on BSSF with $m = 2$ and $F = 250$ for $D_t = 10$. The storage cost of BSSF is half of that of NIX, and is almost same as that of SSF. As for the retrieval cost for the query $T \supseteq Q$, BSSF is comparable to NIX except for $D_q = 1$ and is much superior than SSF. The retrieval cost of BSSF for the query $T \subseteq Q$ is much lower than those of SSF and NIX. The only problem with BSSF is that insertion costs more than SSF and NIX. However, as mentioned before, this cost estimation is based on the worst case assumption. In case of $D_t = 100$, for example, BSSF with $m = 3$ and $F = 2500$ has several desirable properties. In the context of text retrieval, m_{opt}, given by equation (3), is usually used as the value of m. However, our analysis has clarified that we had better set a far smaller value to m of BSSF to facilitate set value accesses.

A further in-depth study on the use of BSSF as a set access facility is going on in our group. The research issues include support of other set operations, cost analysis for cases where the cardinality of target sets varies, and query processing schemes based on BSSF. Research results on these issues will be reported in forthcoming papers.

ACKNOWLEDGEMENTS

The authors would like to thank Mr. Yoshiaki Fukushima, for helpful discussions in study of the false drop probabilities under various conditions. They also thank Prof. Yuzuru Fujiwara and Prof. Isao Suzuki for their encouragement to this research. They appreciate fruitful discussions with many members of the database research laboratory, University of Tsukuba.

APPENDIX A

We present two approximate equations used in the estimations of the false drop probabilities. For any i $(1 \leq i \leq F - m)$, the following equations hold:

$$\begin{aligned}
&\mathrm{Prob}\{b_q^1 = 0 \wedge \cdots \wedge b_q^i = 0\} \\
&= \left(\frac{{}_{F-i}C_m}{{}_FC_m}\right)^{D_q} \\
&= \left(\frac{(F-m)(F-m-1)\cdots(F-m-i+1)}{F(F-1)\cdots(F-i+1)}\right)^{D_q} \\
&= \left(1-\frac{m}{F}\right)^{D_q}\left(1-\frac{m}{F-1}\right)^{D_q}\cdots\left(1-\frac{m}{F-i+1}\right)^{D_q}.
\end{aligned}$$

If $\frac{m}{F-k+1} \ll 1$ is satisfied for $1 \leq k \leq i$,

$$\begin{aligned}
&\approx \left(1-\frac{1}{F}\right)^{mD_q}\left(1-\frac{1}{F-1}\right)^{mD_q}\cdots\left(1-\frac{1}{F-i+1}\right)^{mD_q} \\
&= \left(\frac{(F-1)(F-2)\cdots(F-i)}{F(F-1)\cdots(F-i+1)}\right)^{mD_q} \\
&= \left(1-\frac{i}{F}\right)^{mD_q}.
\end{aligned}$$

Therefore,

$$\mathrm{Prob}\{b_q^1 = 0 \wedge \cdots \wedge b_q^i = 0\} \approx \left(1-\frac{i}{F}\right)^{mD_q}. \qquad (9)$$

APPENDIX B

We derive the retrieval cost of the nested index for $T \subseteq Q$. The number of objects which we must access to answer the query after D_q look-ups of the nested index is given as follows:

$$N\frac{\sum_{i=1}^{D_t-1}\left({}_{D_q}C_i \times {}_{V-D_q}C_{D_t-i}\right)}{{}_VC_{D_t}} + N\frac{{}_{D_q}C_{D_t}}{{}_VC_{D_t}},$$

where the first term expresses the number of objects which do not satisfy the query condition, and the second term expresses the number of objects which actually satisfy the query condition. Therefore,

$$RC = rcD_q + P_uN\frac{\sum_{i=1}^{D_t-1}\left({}_{D_q}C_i \times {}_{V-D_q}C_{D_t-i}\right)}{{}_VC_{D_t}} + P_sN\frac{{}_{D_q}C_{D_t}}{{}_VC_{D_t}}.$$

APPENDIX C

To derive D_q^{opt}, we assume that the number of actual drops is negligible and F_d is very small. This assumption holds in the context of our discussion for $T \subseteq Q$. Then,

$$\begin{aligned}
RC &\approx \left\lceil\frac{N}{Pb}\right\rceil(F-m_q) + F_d(SC_{OID}O_n + P_uN) \\
&= \left\lceil\frac{N}{Pb}\right\rceil Fe^{-\frac{m}{F}D_q} \\
&\quad + (1-e^{-\frac{m}{F}D_q})^{mD_t}(SC_{OID}O_n + P_uN).
\end{aligned}$$

We regard RC as a function of D_q, and differentiate it. D_q^{opt}, the value of D_q which minimizes RC, is given as follows:

$$D_q^{opt} = -\frac{F}{m}\ln\left[1-\left(\frac{\left\lceil\frac{N}{Pb}\right\rceil F}{(SC_{OID}O_n+P_uN)mD_t}\right)^{\frac{1}{mD_t-1}}\right].$$

REFERENCES

[Bert89] E. Bertino and W. Kim. "Indexing Techniques for Queries on Nested Objects," *IEEE Trans. on Knowledge and Data Engineering* 1(2):196–214 June 1989.

[Falo84] C. Faloutsos and S. Christodoulakis. "Signature Files: An Access Method for Documents and Its Analytical Performance Evaluation," *ACM Trans. Office Inf. Syst.* 2(4):267–288 Oct. 1984.

[Falo85] C. Faloutsos. "Access Methods for Text," *ACM Comput. Surv.* 17(1):49–74 Mar. 1985.

[Falo87] C. Faloutsos and S. Christodoulakis. "Description and Performance Analysis of Signature File Methods for Office Filing," *ACM Trans. Office Inf. Syst.* 5(3):237–257 July 1987.

[Falo88] C. Faloutsos and R. Chan. "Fast Text Access Methods for Optical and Large Magnetic Disks: Designs and Performance Comparison," In *Proc. of 14th VLDB Conf.* pp. 280–293 Los Angeles, Calif. 1988.

[Kim88] W. Kim, H.-T. Chou, and J. Banerjee. "Operations and Implementation of Complex Objects," *IEEE Trans. Softw. Eng.* 14(7):985–995 July 1988.

[Kim90] W. Kim. *Introduction to Object-Oriented Databases.* The MIT Press, 1990.

[Pfal80] J. L. Pfaltz, W. J. Berman, and E. M. Cagley. "Partial-Match Retrieval Using Indexed Descriptor Files," *Commun. ACM* 23(9):522–528 Sept. 1980.

[SD87] R. Sacks-Davis, A. Kent, and K. Ramamohanarao. "Multikey Access Methods Based on Superimposed Coding Techniques," *ACM Trans. Database Syst.* 12(4):655–696 Dec. 1987.

[Stei91] J. Stein and D. Maier. "Associative Access Support in GemStone," In K. R. Dittrich, U. Dayal, and A. P. Buchmann, editors, *On Object-Oriented Database Systems* pp. 323–339. Springer-Verlag, 1991.