Analysis of Indexing Schemes to Support Set Retrieval of Nested Objects

Yoshiharu Ishikawa
Graduate School of Information Science
Nara Institute of Science and Technology
8916-5 Takayama, Ikoma, Nara 630-01, Japan
E-mail: ishikawa@is.aist-nara.ac.jp

Hiroyuki Kitagawa
Institute of Information Sciences and Electronics
University of Tsukuba
1-1 Tennoudai, Tsukuba, Ibaraki 305, Japan
E-mail: kitagawais.tsukuba.ac.jp

Abstract

Efficient retrieval of nested objects is an important issue in advanced database systems. So far, many indexing methods for nested objects are proposed. However, they do not consider retrieval of nested objects based on the set comparison operators such as $\subseteq$ and $\subsetneq$. In this paper, we propose four set access facilities for nested objects and compare their performance in terms of retrieval cost, storage cost, and update cost. Our analysis shows that a combination of the signature file method and the nested index is very promising for set retrieval of nested objects.

1 Introduction

Nested objects frequently appear in databases for advanced application areas. Set values appear as attributes of nested objects and many advanced database systems support some kind of constructs that can express and manipulate set values. Therefore, efficient indexing methods are required to facilitate retrieval of nested objects based on set comparison operators such as $\subseteq$ and $\subsetneq$. In order to support efficient retrieval of nested objects, new indexing methods such as the nested index, the path index, and the multi-index have been proposed [1, 2]. However, they are not designed to support such set retrieval of nested objects. We have proposed the use of superimposed coded signature files as efficient set retrieval facilities for non-nested objects with set-valued attributes and showed their potential capabilities [8].

In this paper, we extend the target to multi-level nested objects with set-valued attributes and investigate efficient indexing methods for them. We consider four candidate set access facilities combining the signature file method and the nested index, and compare them in terms of retrieval cost, storage cost, and update cost.

The reminder of this paper is organized as follows. Section 2 introduces the notion of set objects and set retrieval of nested objects. Section 3 explains the four set access facilities for nested objects. Section 4 describes our cost model. Section 5 shows the results of our analysis of the four access facilities. Section 6 gives a summary and concludes the paper.

2 Set Retrieval of Nested Objects

In this section, we informally define the notion of nested objects as the basis of our discussion. Next, the concept of set retrieval of nested objects is illustrated and a sample query is shown.

2.1 Nested Objects

An object is a tuple-structured data defined by the tuple constructor $\{\ldots\}$ and has one or more attributes. Each object is identified by its object identifier (OID). The structure of objects in a class is specified by the class definition. A set of class definitions is called a schema. There are two types of attributes: an atomic attribute takes a primitive value or an OID as its value, and a set attribute takes a set of primitive values or an OID set of objects in some class as its value. In a schema, a set attribute is specified by the set constructor $\{\ldots\}$. An example schema is shown in Figure 1.

If an object $O$ has an OID of some object $O'$ as a primitive attribute value, or has an OID of some object $O'$ in its set attribute value, we say that the object $O$ refers to the object $O'$. Next, assume that...
\{Emp = [ename: str, proj: Proj, hobbies: str, ...],
    Proj = [name: str, dept: Dept, emp: Emp, ...],
    Dept = [name: str, items: set(str), ...]\}

Figure 1: An Example Schema

classes \(C_1, C_2, \ldots, C_n\) are defined in a schema. A path \(P\) is defined as \(P = C_1.A_1.A_2.\ldots.A_n\), where \(A_i\) (\(1 \leq i \leq n - 1\)) is an attribute of the class \(C_i\) and takes an OID of a \(C_{i+1}\) object or an OID set of \(C_{i+1}\) objects as its value. \(A_n\) is an attribute of \(C_n\), and can take a primitive value, a set of primitive values, an OID, or an OID set as its value. An instance of the path \(P\) has the form \(O_1.O_2.\ldots.O_n.X\), where \(O_1\) is an OID of a \(C_1\) object (this object is called the root object). If the attribute \(A_i\) (\(1 \leq i \leq n - 1\)) is a primitive attribute, \(O_i+1\) is an OID of a \(C_{i+1}\) object which appears as the \(A_i\) value of \(O_i\). If \(A_i\) is a set attribute, the \(A_i\) value of \(O_i\) contains the OID \(O_{i+1}\) as a set element. \(X\) is the \(A_n\) value of the object \(O_n\).

2.2 Sample Query

In this paper, we consider the following query form over a path \(C_1.A_1.A_2.\ldots.A_n\):

select \{attribute value(s) of \(C_1\)\}
from \(C_1\)
where \(A_1.A_2.\ldots.A_n\ op \{set value\},\)
where \(A_1.A_2.\ldots.A_{n-1}\) are primitive attributes and \(A_n\) is a set attribute. The comparison operator \(op\) can be \(\geq\) or \(\subseteq\). (set value) is called the query set \(Q\), and each set stored as an \(A_n\) value in the database and queried is called a target set \(T\). An example of such query is the following query \(Q_1\) based on the schema in Figure 1:

\[Q_1: select \text{ename}\]
\[ from \text{Emp}\]
\[ where \text{proj.dept.items} \geq \{"orange", "banana"\}\]

This kind of query is called \(T \geq Q\) (has-subset). If the comparison operator is \(\subseteq\), the query is called \(T \subseteq Q\) (is-subset).

3 Set Access Facilities for Nested Objects

In this section, we introduce four set access facilities for nested objects: \(\text{InBSS}, \text{InIX}, \text{InBSS\_IX}, \text{and InIX\_IX}\). As a preparation, first we introduce the notion of a signature file as a set access facility.

3.1 Signature File as a Set Access Facility

Signature files were originally designed for text retrieval [3, 4, 5, 7, 10]. We have proposed the use of signature files as efficient set retrieval facilities and showed their potential capabilities for non-nested objects [8]. For set retrieval, a target signature is generated for each set element by hashing. Every element signature has \(F\) bit length, and \(m\) bits are set to "1". Then, a target signature is obtained by bit-wise OR-ing (superimposed coding) element signatures of all the elements in the target set and stored in the signature file with the corresponding OID. Figure 2 shows the generation of the target signature for a target set \{"orange", "banana", "grape"\}.

\[
\begin{array}{c|c}
\text{element} & \text{element signature} \\
\hline
\text{"orange"} & 0100000000100000 \\
\text{"banana"} & 0000000100000000 \\
\text{"grape"} & 0000000101000000 \\
\end{array}
\]

Figure 2: Generation of a Target Signature

\((F = 16, m = 2)\)

When a query is given, the query signature is generated from the query set in the same way as target signatures. Then, each target signature in the signature file is examined over the query signature for potential match. If the target signature satisfies a predefined condition implied by the query condition, the corresponding data object becomes a candidate that may satisfies the query. Such a data object is called a drop. Each target set becomes a drop if the following condition is satisfied [9]:

\[T \geq Q; \text{query signature} \land \text{target signature} = \text{query signature}\]

\[T \subseteq Q; \text{query signature} \land \text{target signature} = \text{target signature}\]

where \(\land\) stands for a bit-wise AND operation. The last step is called false drop resolution, and 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.

There are a number of choices in physical signature file organizations [5]. In this paper, we use the bit-sliced signature file (BSSF), a well-known storage organization for signature files. BSSF stores signatures in a column-wise manner. Thus \(F\) files (called bit-slice files) are created. Figure 3 illustrates the file structure of BSSF. The result in [8] says that BSSF is promising as a set access facility for non-nested objects.

3.2 Set Access Facilities for Nested Objects

Now we introduce four set access facilities for nested objects. \(\text{InBSS}\) is an extension of BSSF to set accesses of nested objects. \(\text{InIX}\) is based on the nested index
1. BSSF is searched based on the query condition and an OID set of \( C_1 \) objects is obtained.
2. For each element of the OID set found in step 1, a forward traversal \([1, 2]\) is performed. Thus \( C_n \) objects are retrieved.
3. Each \( C_n \) object is examined whether it actually satisfies the query condition. If it satisfies the condition, the corresponding \( C_1 \) object is returned as a query result.

\( I_{\text{NIX}} \):

1. For each element in the query set, \( NIX \) is searched. Thus \( D_q \) OID sets of \( C_1 \) objects are obtained, where \( D_q \) is the cardinality of the query set.
2. For \( T \supseteq Q \), the intersection of the \( D_q \) sets is taken. For \( T \subseteq Q \), the union is taken.
3. For \( T \supseteq Q \), \( C_1 \) objects are retrieved based on the OID set and returned.

\( 3' \). For \( T \subseteq Q \), the following process is done:

(a) Forward traversals are performed based on the OID set, and the corresponding \( C_n \) objects are checked whether they actually satisfy the query condition.
(b) The root \( C_1 \) objects of the \( C_n \) objects which satisfy the condition (a) are returned.

\( I_{\text{BSSF}, \text{NIX}} \):

Both \( T \supseteq Q \) and \( T \subseteq Q \) are processed as follows:

1. BSSF is searched based on the query condition and an OID set of \( C_1 \) objects is obtained.
2. For each element of the OID set, the corresponding \( C_n \) object is retrieved and checked whether it actually satisfies the query condition.
3. For each \( C_n \) object that satisfies the condition, NIX is searched using its OID as a key value. As a result, an OID set of \( C_1 \) objects is obtained.
4. \( C_1 \) objects are retrieved based on the OID set and returned.

\( I_{\text{NIX}, \text{NIX}} \):

1. For each element in the query set, \( NIX_1 \) is searched. Thus \( D_q \) OID sets of \( C_1 \) objects are obtained.
2. For \( T \supseteq Q \), the intersection of the \( D_q \) sets is taken. For \( T \subseteq Q \), the union is taken.
3. For \( T \supseteq Q \), the following process is done:

(a) For each element of the OID set, \( NIX_2 \) is searched. Then, an OID set of \( C_1 \) objects is obtained.
(b) \( C_1 \) objects are retrieved based on the OID set and returned.

\( 3' \). For \( T \subseteq Q \), the following process is done:

(a) \( C_n \) objects are retrieved based on the OID set and examined whether they actually sat-
isfy the query condition.
(b) NIX2 is searched using the OIDs of \( C_n \) objects that satisfy the condition (a) and an OID set of \( C_1 \) objects is obtained.
(c) \( C_1 \) objects are retrieved based on the OID set and returned.

3.4 Update Algorithms

Algorithms for inserting a new path instance \( P = C_1.A_1.A_2.\cdots.A_n \) are already described in Subsection 3.2. Therefore, here we only show the deletion algorithms. Suppose that a \( C_n \) object is deleted from the database and that there are no backward references to \( C_1 \) objects. In this case, the following deletion algorithms can be used.

**Inss:**
1. A query signature is generated from the set in the \( C_n \) object, and BSSF is searched based on the set equality condition \( (T \equiv Q) \). Namely, the target signatures which are completely coincident with the query signature are searched by scanning \( F \) bit-slice files. Thus an OID set of candidate \( C_1 \) objects is obtained.
2. For each OID in the OID set, a forward traversal is performed to check whether it actually refers the \( C_n \) object.
3. For each OID that satisfies the above condition, the corresponding entry is deleted from the BSSF file.

**INX1:**
1. NIX is searched based on the set equality condition \( (T \equiv Q) \). Namely, for each element in the set in the \( C_n \) object, the corresponding NIX entry is retrieved. Thus \( D_0 \) OID sets of \( C_1 \) objects are obtained.
2. The intersection of the \( D_0 \) sets is taken. Thus an OID set of \( C_1 \) objects is obtained.
3. For each OID in the OID set, a forward traversal is performed to check whether it actually refers the \( C_n \) object.
4. For the OIDs that satisfies the above condition, the corresponding entries are deleted from the NIX file.

**Inss-nix and Inx-nix:** Algorithms for **Inss-nix** and **Inx-nix** are straightforward. For **Inss-nix**, the entry of the BSSF file that corresponds to the \( C_n \) object is deleted and the entries of the NIX file that refer the \( C_n \) object are deleted. Deletions on \( \text{Inss-nix} \) are processed in a similar manner.

4 Cost Model

In this section, our cost model for four set access facilities for nested objects is developed. First, as a preliminary, cost formulas for BSSF and NIX are shown. Then, retrieval costs, storage costs, and update costs are shown. Costs are derived in terms of the number of page accesses.

4.1 Cost Formulas for BSSF and NIX

Before describing our cost model for set access facilities for nested objects, general cost formulas for BSSF and NIX are shown. They are revised versions of [8]. Symbols and their definitions are shown in Table 1.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition and Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( N )</td>
<td>total number of objects</td>
</tr>
<tr>
<td>( V )</td>
<td>cardinality of the set domain of the attribute ( A_n ) ((= 10,000))</td>
</tr>
<tr>
<td>( b )</td>
<td>size of a disk page ((= 4096 \text{ bytes}))</td>
</tr>
<tr>
<td>( oid )</td>
<td>number of bits per byte ((= 8))</td>
</tr>
<tr>
<td>( P_o )</td>
<td>size of an OID ((= 8 \text{ bytes}))</td>
</tr>
<tr>
<td>( Fd[i] )</td>
<td>number of page accesses to fetch an object ((= 1))</td>
</tr>
<tr>
<td>( S_{\text{bit}}(N) )</td>
<td>storage cost for a bit-slice file ((= \frac{N}{M}))</td>
</tr>
<tr>
<td>( M[i] )</td>
<td>number of bit-slice files to be retrieved</td>
</tr>
<tr>
<td>( L_{\text{OIDS}}(N) )</td>
<td>access cost for the OID file ((= \frac{P + oid}{512}))</td>
</tr>
<tr>
<td>( O_n )</td>
<td>number of OIDs in a disk page ((= \frac{P}{oid}))</td>
</tr>
<tr>
<td>( S_{\text{OID}}(N) )</td>
<td>size of the OID file ((= \frac{N}{O_n})) pages</td>
</tr>
<tr>
<td>( \alpha(i)(N) )</td>
<td>number of actual drops per page of the OID file ((= \frac{A_i(N)}{S_{\text{OID}}(N)}))</td>
</tr>
<tr>
<td>( \beta(y) )</td>
<td>size of a leaf node index entry of NIX</td>
</tr>
<tr>
<td>( lpt(x, y) )</td>
<td>total number of leaf pages</td>
</tr>
<tr>
<td>( nlp(x, y) )</td>
<td>total number of non-leaf pages</td>
</tr>
<tr>
<td>( kl )</td>
<td>size of a key value ((= 8 \text{ bytes}))</td>
</tr>
<tr>
<td>( ol )</td>
<td>size of a field which specifies the number of OID entries ((= 2 \text{ bytes}))</td>
</tr>
<tr>
<td>( f )</td>
<td>average fanout from a non-leaf node ((= 218))</td>
</tr>
</tbody>
</table>

\(^{1}\)In general case such that \( A_i \) \((1 \leq i \leq n - 1)\) may be a set attribute, slightly different algorithm is required. However, we omit the description here.
BSSF  The retrieval cost of BSSF, $RC_{BSSF}(c)(N)$, 
is given in a general form [8]:

$$RC_{BSSF}(c)(N) = SC_{Sat}(N) \times M(c) + LC_{OD}(c)(N) + F_d(A(c)(N) + F_d(N - A(c)(N)))$$

(1)

$$LC_{OD}(c)(N) = SC_{OD}(N) \times \min(Fd(c)|O_h - a(c)(N)| + a(c)(N), 1)$$

(2)

where $c$ denotes the type of the query ($T \supseteq Q, T \subseteq Q$).

Formulas for false drop probability $Fd(c)$ are shown below. $M(c)$ is the number of bit-slice files to be accessed and given as follows [8]:

$$M(T \supseteq Q) = m_q$$

(3)

$$M(T \subseteq Q) = F - m_q,$$

(4)

where $m_q$ is the expected number of “1”’s (weight) in the query signature and given by $m_q \approx F(1 - e^{-\frac{P_t}{D_t}})$.

The storage cost of BSSF is simply derived as

$$SC_{BSSF}(N) = SC_{Sat}(N) \times F + SC_{OD}(N).$$

(5)

The update algorithms used in this paper are different from [8] and more efficient ones. Let us consider the insertion cost of BSSF. The weight of the target signature to be inserted is given by $m_t \approx F(1 - e^{-\frac{P_t}{D_t}})$. To insert the signature, $m_t$ bit-slice files are accessed to turn the bits corresponding to the insert position to “1”. Moreover, the OID file is accessed to insert the OID of the data object. Therefore, the insertion cost $IC_{BSSF}$ is

$$IC_{BSSF} = 2(m_t + 1).$$

(6)

2 is multiplied because of the read cost and the write cost. When deleting an entry from BSSF, an OID is given to specify the entry. First, the OID file is accessed and the bit position corresponding to the OID is determined by linear search. Second, the bit positions of $m_t$ bit-slice files are reversed to “0”’s. Therefore, the deletion cost $DC_{BSSF}(N)$ is given as

$$DC_{BSSF}(N) = \frac{SC_{OD}(N)}{2} + 2m_t + 1.$$

(7)

The third term is the write cost for the OID file.

NIX  Let $r(x, y)$ be the search cost of NIX for a key value, where $x$ is the total number of keys in the NIX file and $y$ is the average number of OIDs corresponding to a key value. In [8], we derived the cost formulas for NIX as a set access facility based on the cost model in [2]. Here we do not re-derive the cost formulas due to the space limitation. Detailed descriptions are found in [2, 8]. The insert and delete costs are written as follows:

$$IC_{NIX}(x, y) = DC_{NIX}(x, y) = (rc(x, y) + 1) \times D_q.$$  

(8)

In this paper, we do not include the costs of index page splits/shrinks, and only consider the cost of leaf-page update.

False Drop Probabilities and Actual Drops False drop probability is an important measure to estimate the performance of signature files and given by the following formula [6]:

$$Fd = \frac{\text{false drops}}{\text{total number of objects - actual drops}}.$$  

In this paper, we use the following estimations [9]:

$$Fd(T \supseteq Q) \approx (1 - e^{-\frac{P_t}{D_t}})^{m_t}$$

(9)

$$Fd(T \subseteq Q) \approx (1 - e^{-\frac{P_t}{D_t}})^n$$

(10)

where $D_t$ and $D_q$ are the cardinalities of the target set and the query set, respectively. Actual drops $A(c)(N)$ are given as follows [8]:

$$A(T \supseteq Q)(N) = N^{\frac{m_t - D_t}{C_{BSSF}} - \frac{m_t - D_t}{C_{OA}}}$$

(11)

$$A(T \subseteq Q)(N) = N^{\frac{D_t - D_q}{C_{BSSF}}},$$

(12)

4.2 Retrieval Costs

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition and Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$PI$</td>
<td>total number of the path instances</td>
</tr>
<tr>
<td>$N_i$</td>
<td>number of the $C_i$ objects</td>
</tr>
<tr>
<td>$FT(z)$</td>
<td>forward traversal cost ($z$ is the number of the root objects)</td>
</tr>
</tbody>
</table>

We evaluate retrieval costs of four set access facilities for queries over the path $C_1, A_1, A_2, \ldots, A_n$ described in Subsection 2.2. Table 2 shows the symbols used in the cost formulas. To derive retrieval costs, we make two assumptions: (1) $A_1, A_2, \ldots, A_{n-1}$ are primitive attributes and $A_n$ is a set attribute as stated in Subsection 2.2. (2) The value of $A_i$ ($1 \leq i \leq n - 1$) is not NULL and there is no reference sharing. Thus the number of instances of the path $P$ becomes $PI = N_1 (= N_2 = \cdots = N_n)$. Next, we consider the forward traversal cost. If $x$ OIDs are given and forward traversals are performed to retrieve corresponding $C_{n-1}$ OIDs...
objects, the cost is simply given as \( FT(x) = x \times n \) under the above assumption 4.

Retrieval cost formulas for four set access facilities are shown below.

\[ I_{BSSF} \] Based on the formulas in Subsection 4.1, the retrieval cost of \( I_{BSSF} \) is given as:

\[
RC\{I_{BSSF}, c\} = SC_{BSSF}(P1) + M(c) + LC_{OID}(c)(P1) + FT(A(c))\times D_q + PD\times A[T\geq Q](N_n).
\]

\[ I_{NIX} \] The retrieval costs of \( I_{NIX} \) depend on the query, and are given as follows:

\[ RC\{I_{NIX}, T \geq Q\} = r(c) \times (V, \frac{D_q N_n}{C}) \times D_q + PD \times A[T\geq Q](N_n) \]  

\[ RC\{I_{NIX}, T \subseteq Q\} = r(c) \times (V, \frac{D_q N_n}{C}) \times D_q + \]

\[
FT\left(\frac{\sum_{i=1}^{n-1} \sum_{j=1}^{n} C_{ij} \times \frac{D_q N_n}{C}}{V \times C_n}\right).\]

\[ I_{BSSF-NIX} \]

\[ RC\{I_{BSSF-NIX}, c\} = RC_{BSSF}(c)(N_n) + r(c)(N_n, 1) \times A(c)(N_n) + PD\times A(c)(N_n) \]

\[ I_{NIX-NIX} \] The retrieval costs of \( I_{NIX-NIX} \) also depend on the query:

\[ RC\{I_{NIX-NIX}, T \geq Q\} = r(c) \times (V, \frac{D_q N_n}{C}) \times D_q + r(c)(N_n, 1) \times A[T\geq Q](N_n) + PD\times A[T\geq Q](N_n) \]

\[ RC\{I_{NIX-NIX}, T \subseteq Q\} = r(c) \times (V, \frac{D_q N_n}{C}) \times D_q + \]

\[
\sum_{i=1}^{n-1} \sum_{j=1}^{n} C_{ij} \times \frac{D_q N_n}{C} \times A[T\subseteq Q](N_n) + PD\times A[T\subseteq Q](N_n).\]

4.3 Storage Costs

Storage costs for four set access facilities are as follows:

\[ SC\{I_{BSSF}\} = SC_{BSSF}(P1) \]  

\[ SC\{I_{NIX}\} = SC_{NIX}(V, \frac{D_q N_n}{C}) \]  

\[ SC\{I_{BSSF-NIX}\} = SC_{BSSF}(N_n) + SC_{NIX}(N_n, 1) \]

\[ SC\{I_{NIX-NIX}\} = SC_{NIX}(V, \frac{D_q N_n}{C}) + SC_{NIX}(N_n, 1) \]

where \( SC_{NIX}(x, y) \) is the storage cost of NIX when \( x \) is the number of keys and \( y \) is the average number of OIDs corresponding to a key value. These cost formulas are shown in [8].

4.4 Update Costs

Insertion costs are based on the algorithms in Subsection 3.2:

\[ IC\{I_{BSSF}\} = IC_{BSSF} \]  

\[ IC\{I_{NIX}\} = IC_{NIX}(V, \frac{D_q N_n}{C}) \]

\[ IC\{I_{BSSF-NIX}\} = IC_{BSSF} + r(c)(N_n, 1) + 1 \]

\[ IC\{I_{NIX-NIX}\} = IC_{NIX}(V, \frac{D_q N_n}{C}) + r(c)(N_n, 1) + 1 \]

Deletion costs are as follows:

\[ DC\{I_{BSSF}\} = SC_{BSSF}(P1) \times F + LC_{OID}(T=q)(PI) + FT(1 + Fd[T\geq Q](N_n - 1)) + 2(m_{i+1}) \]

\[ DC\{I_{NIX}\} = r(c) \times (V, \frac{D_q N_n}{C}) \times D_q + \]

\[ FT\left(\max\left(\frac{N_c}{v \times D_q}, 1\right)\right) + r(c) \times (V, \frac{D_q N_n}{C}) + 1 \]

\[ DC\{I_{BSSF-NIX}\} = DC_{BSSF}(N_n) + r(c)(N_n, 1) + 1 \]

\[ DC\{I_{NIX-NIX}\} = DC_{NIX}(V, \frac{D_q N_n}{C}) + r(c)(N_n, 1) + 1 \]

To compute formula (28), \( Fd[T\geq Q] \) is required. It is derived in [9].

5 Cost Analysis

Before comparing the retrieval costs of four set access facilities, we describe the parameter settings. \( N_i \), the number of objects in the class \( C_i \), is set to \( N_1 = N_2 = \cdots = N_{10} = 30,000 \). As the cardinality of target sets \( D_q \), we consider two cases: \( D_q = 10 \) and \( D_q = 100 \), and for the length of the path, we compare three cases \( n = 2, 3, 4 \). For \( I_{BSSF} \) and \( I_{BSSF-NIX} \), it is necessary to set the BSSF parameters. We follow the following policy: (1) The storage costs of \( I_{BSSF} \) and \( I_{BSSF-NIX} \) are at most equal to that of \( I_{NIX} \) and \( I_{NIX-NIX} \). This policy restricts the signature size \( F \). When \( D_q = 10 \), we use \( F = 500 \) (bits), and when \( D_q = 100 \), \( F = 5000 \) (bits) is used. (2) The parameter \( m \) is set to \( m = 2 \) based on the results in [8].

5.1 Retrieval Costs

The representative retrieval costs for \( T \geq Q \) are shown in Figure 4 (\( D_q = 10 \)) and Figure 5 (\( D_q = 100 \)). Two figures have a similar tendency. For \( D_q = 1 \), \( I_{BSSF} \) gives the worst cost. This is because \( I_{BSSF} \) requires forward traversals to process the query. When \( D_q = 1 \), actual drops and false drops exist so that the overhead of the forward traversal cost affects the overall cost. However, when \( D_q \geq 2 \), drops are almost negligible, so the retrieval costs increase linearly. The change of the path length \( n \) does not affect the retrieval costs very well because the numbers of drops are small except for \( D_q = 1 \).
Figure 4: Retrieval Cost \( T \geq Q, D_t = 10, n = 3 \)

Figure 5: Retrieval Cost \( T \geq Q, D_t = 100, n = 3 \)

The representative retrieval costs for \( T \subseteq Q \) are shown in Figure 6 \( (D_t = 10) \) and Figure 7 \( (D_t = 100) \). \( I_{\text{BSSF}} \) and \( I_{\text{BSSF-MIX}} \) have almost the same costs and so do \( I_{\text{INIX}} \) and \( I_{\text{INIX-MIX}} \). When \( D_t = 10 \) and \( D_q \) is very small, the retrieval costs of \( I_{\text{BSSF}} \) and \( I_{\text{BSSF-MIX}} \) are larger than those of \( I_{\text{INIX}} \) and \( I_{\text{INIX-MIX}} \). But \( I_{\text{BSSF}} \) and \( I_{\text{BSSF-MIX}} \) are generally better in the other cases. Furthermore, the retrieval costs of \( I_{\text{BSSF}} \) and \( I_{\text{BSSF-MIX}} \) can be improved by using the smart retrieval strategy proposed in [8]. Therefore, for \( T \subseteq Q \), \( I_{\text{BSSF}} \) and \( I_{\text{BSSF-MIX}} \) are considered to be superior to \( I_{\text{INIX}} \) and \( I_{\text{INIX-MIX}} \). As it is not shown in Figures 6 and 7, the cost of \( I_{\text{BSSF}} \) drastically increases for very large \( D_t \) values. Therefore, \( I_{\text{BSSF-MIX}} \) is the most efficient and stable one for \( T \subseteq Q \).

5.2 Storage Costs

<table>
<thead>
<tr>
<th>Index</th>
<th>Storage Cost (pages)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( I_{\text{BSSF}} )</td>
<td>559</td>
</tr>
<tr>
<td>( I_{\text{INIX}} )</td>
<td>629</td>
</tr>
<tr>
<td>( I_{\text{BSSF-MIX}} )</td>
<td>693</td>
</tr>
<tr>
<td>( I_{\text{INIX-MIX}} )</td>
<td>763</td>
</tr>
</tbody>
</table>

Table 4: Storage Cost \( (D_t = 100, F = 5000) \)

<table>
<thead>
<tr>
<th>Index</th>
<th>Storage Cost (pages)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( I_{\text{BSSF}} )</td>
<td>5059</td>
</tr>
<tr>
<td>( I_{\text{INIX}} )</td>
<td>10047</td>
</tr>
<tr>
<td>( I_{\text{BSSF-MIX}} )</td>
<td>5193</td>
</tr>
<tr>
<td>( I_{\text{INIX-MIX}} )</td>
<td>10181</td>
</tr>
</tbody>
</table>

Storage costs are shown in Tables 3 and 4. When \( D_t = 10 \), the storage costs are almost in the same degree. When \( D_t = 100 \), the storage costs of \( I_{\text{BSSF}} \) and \( I_{\text{BSSF-MIX}} \) are half of those of \( I_{\text{INIX}} \) and \( I_{\text{INIX-MIX}} \).

5.3 Update Costs

Insertion costs are shown in Table 5 \( (D_t = 10, F = 500) \) and Table 6 \( (D_t = 100, F = 5000) \). Four access facilities have almost the same costs.

Deletion costs are shown in Table 7 \( (D_t = 10, F = 500) \) and Table 8 \( (D_t = 100, F = 5000) \). The cost of \( I_{\text{BSSF}} \) is prohibitively larger than other access facilities. This is because all bit-slices of the BSSF file are retrieved to check the query condition \( T \equiv Q \). We may be able to improve the cost by the smart retrieval strategy [8]. Other three facilities are almost in the same degree.
<table>
<thead>
<tr>
<th>Table 5: Insertion Cost ($D_i = 10, F = 500$)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Index</strong></td>
</tr>
<tr>
<td>$I_{ESF}$</td>
</tr>
<tr>
<td>$I_{SN}$</td>
</tr>
<tr>
<td>$I_{ESF_{SN}}$</td>
</tr>
<tr>
<td>$I_{SN_{SN}}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 6: Insertion Cost ($D_i = 100, F = 5000$)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Index</strong></td>
</tr>
<tr>
<td>$I_{ESF}$</td>
</tr>
<tr>
<td>$I_{SN}$</td>
</tr>
<tr>
<td>$I_{ESF_{SN}}$</td>
</tr>
<tr>
<td>$I_{SN_{SN}}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 7: Deletion Cost ($D_i = 10, F = 500, n = 3$)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Index</strong></td>
</tr>
<tr>
<td>$I_{ESF}$</td>
</tr>
<tr>
<td>$I_{SN}$</td>
</tr>
<tr>
<td>$I_{ESF_{SN}}$</td>
</tr>
<tr>
<td>$I_{SN_{SN}}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 8: Deletion Cost ($D_i = 100, F = 5000, n = 3$)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Index</strong></td>
</tr>
<tr>
<td>$I_{ESF}$</td>
</tr>
<tr>
<td>$I_{SN}$</td>
</tr>
<tr>
<td>$I_{ESF_{SN}}$</td>
</tr>
<tr>
<td>$I_{SN_{SN}}$</td>
</tr>
</tbody>
</table>

6 Summary and Conclusion

In this paper, we have proposed four set access facilities $I_{ESF}$, $I_{SN}$, $I_{ESF_{SN}}$, and $I_{SN_{SN}}$ for nested objects and compared their performance. We developed a cost model and analyzed the retrieval costs for two queries ($T \supseteq Q$, $T \subseteq Q$) and the storage and update costs.

As for the retrieval cost for $T \supseteq Q$, the analysis show that four access facilities have similar performance except for $D_q = 1$. When $D_q = 1$, $I_{ESF}$ is the worst and $I_{SN}$ is the best. However, as for the retrieval cost for $T \subseteq Q$, $I_{ESF}$ and $I_{ESF_{SN}}$ show relatively stable performance and are better than $I_{SN}$ and $I_{SN_{SN}}$ for reasonable range of $D_q$ values. For very large $D_q$ values, $I_{ESF_{SN}}$ is superior to $I_{ESF}$.

The storage costs of $I_{ESF}$ and $I_{ESF_{SN}}$ are equal to or smaller than those of $I_{SN}$ and $I_{SN_{SN}}$. As for update costs, four set access facilities are almost in the same order except the deletion cost of $I_{ESF}$. However, we may be able to improve the deletion cost of $I_{ESF}$ by the smart retrieval strategy.

From our analysis, we can conclude that if we must select only one access facility from the four candidates, it is better to use $I_{ESF_{SN}}$ because of its stable performance and smaller storage cost. If the case of $T \supseteq Q$ and $D_q = 1$ is important, $I_{SN}$ becomes another candidate. But it cannot support $T \subseteq Q$ queries very well.

Further study of set access facilities for nested objects is going on in our group. The research issues include: (1) processing of another type of query (e.g., set equality), (2) applicability of the smart retrieval strategy, and (3) cost analysis in other configurations of nested objects (e.g., when a path contains a set attribute in the middle of the path).

Acknowledgement

This work was partially supported by the Grant-in-Aid for Scientific Research from the Ministry of Education, Science, and Culture of Japan.

References


