IEICE TRANS. INF. & SYST., VOL. E78-D, NO. 11 NOVEMBER 1995

PAPER *Special Issue on Advanced Database Technologies*

Design and Performance Analysis of Indexing Schemes for Set Retrieval of Nested Objects

Yoshiharu ISHIKAWA† *and* **Hiroyuki KITAGAWA**††, *Members*

SUMMARY Efficient retrieval of nested objects is an important issue in advanced database systems. So far, a number of indexing methods for nested objects have been proposed. However, they do not consider retrieval of nested objects based on the set comparison operators such as $\supseteq$ and $\subseteq$. Previously, we proposed four set access facilities for nested objects and compared their performance in terms of retrieval cost, storage cost, and update cost. In this paper, we extend the study and present refined algorithms and cost formulas applicable to more generalized situations. Our cost models and analysis not only contribute to the study of set-valued retrieval but also to cost estimation of various indexing methods for nested objects in general.

key words: *set retrieval, nested object, signature file, nested index, access method, performance analysis*

1. Introduction

Nested objects frequently appear in databases for advanced application areas. Many advanced database systems support some kind of construct to express and manipulate set values. Therefore, efficient indexing methods are required to facilitate retrieval of nested objects based on set comparison operators such as $\supseteq$. In order to support efficient retrieval of nested objects, several 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 set-valued retrieval of nested objects. We have proposed the use of superimposed coded *signature files* as efficient set access facilities for non-nested objects with set attributes and showed their potential capabilities [8]. Previously, we extended the target to multilevel nested objects with set attributes and proposed four set access facilities [9]. Their performance was compared in terms of retrieval, storage, and update costs, but their estimation was performed under the assumption that nested objects do not have set attributes except for leaf-level attributes.

In this paper, we consider more general situations in which nested objects may have set attributes in their nonleaf-level attributes. We present revised algorithms and cost formulas taking this extension into account and show cost evaluation results. So far, most performance analyses of indexes for nested objects have been performed assuming that nested objects do not have set attributes [1], [2]. Therefore, some of our results also contribute to analysis of indexes for nested objects in general as well as their set-valued retrieval.

The remainder 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. Preliminaries

In this section, we informally define the notion of nested objects as the basis of the following discussion. Then a sample query is shown.

An *object* comprises tuple-structured data defined by the *tuple constructor* ([...]) 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.* We consider 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* ({...}). An example schema is shown in Fig. 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 *references* object O'. Next, assume that classes $C_1, C_2, \ldots, C_n$ are defined in a schema. A *path* P is defined as $P = C_1.A_1.A_2.\cdots.A_n$, 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

```
{Dept = [dname:str, projs:{Proj}, ...],
 Proj = [pname:str, emps:{Emp}, leader:Emp, ...],
 Emp  = [ename:str, hobbies:{str}, ...]}
```

Fig. 1 An example schema.

Manuscript received December 20, 1994.
Manuscript revised May 2, 1995.

†The author is with the Graduate Institute of Information Science, Nara Institute of Science and Technology, Ikoma-shi, 630-01 Japan.
††The author is with the Institute of Information Sciences and Electronics, University of Tsukuba, Tsukuba-shi, 305 Japan.

instance of the path P has the form $O_1.O_2.\cdots.O_n.X$, 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 OID O_{i+1} as a set element. X is the A_n value of the object O_n.

In this paper, we consider the following query form over the path $C_1.A_1.A_2.\cdots.A_n$:

> **select** ⟨attribute value(s) of C_1⟩
> **from** C_1
> **where** $A_1.A_2.\cdots.A_n$ ⟨op⟩ ⟨set value⟩,

where $A_1, A_2, \ldots, A_{n-1}$ are primitive or set attributes and A_n is a set attribute. The comparison operator ⟨op⟩ can be $\supseteq$ or $\subseteq$. ⟨set value⟩ is called the *query set* (Q), and each set stored as an A_n value in the database is called a *target set* (T). An example of such a query is the following query $\mathbf{Q}_1$ based on the schema in Fig. 1.

$\mathbf{Q}_1$:
> **select** dname
> **from** Dept
> **where** projs.emps.hobbies $\supseteq$ {"baseball", "skiing"}

$\mathbf{Q}_1$ retrieves department names such that hobbies of some employees in the department's projects include both "baseball"and "skiing". This kind of query is called $T \supseteq Q$ (*has-subset*). If the comparison operator is $\subseteq$, the query is called $T \subseteq Q$ (*is-subset*).

In [9], we restricted $A_1, A_2, \ldots, A_{n-1}$ to primitive attributes. However, in this paper, we relax this restriction and deal with more practical situations. Therefore, algorithms and cost models in the following sections are extended and revised.

3. Set Access Facilities for Nested Objects

In this section, we introduce four set access facilities for nested objects: $\mathcal{I}_{\text{BSSF}}$, $\mathcal{I}_{\text{NIX}}$, $\mathcal{I}_{\text{BSSF-NIX}}$, and $\mathcal{I}_{\text{NIX-NIX}}$. As a preparation, we first 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 proposed and used in the text retrieval area [5], [7], [12]. 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 target set and stored in the signature file. First, an *element 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 bitwise 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 {"baseball", "skiing", "golf"}.

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 a potential match. If the target signature satisfies a predefined condition implied by the query condition, the corresponding data object becomes a candidate that may satisfy the query. Such a data object is called a *drop*. Each target set becomes a drop if the following conditions are satisfied [8], [11]:

$T \supseteq Q$: *query signature* $\wedge$ *target signature* $=$ *query signature*

$T \subseteq Q$: *query signature* $\wedge$ *target signature* $=$ *target signature*,

where '$\wedge$' stands for a bit-wise AND operation. The last step of query processing is called *false drop resolution*, and each drop is accessed and examined as to 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 [7]. In this study, we use the *bit-sliced signature file* (*BSSF*), a well-known storage organization for signature files. BSSF stores signatures in a columnar manner. Thus F files (they are called *bit-slice files*) are created. Figure 3 illustrates the file structure of BSSF. The result in [8] indicates that BSSF is promising as a set access facility for non-nested objects.

Element	Element Signature
"baseball"	0100000000010000
"skiing"	0000000100000100
"golf"	0000000100100000
Target Signature →	0100000100110100

Fig. 2 Generation of a target signature. ($F = 16$, $m = 2$)

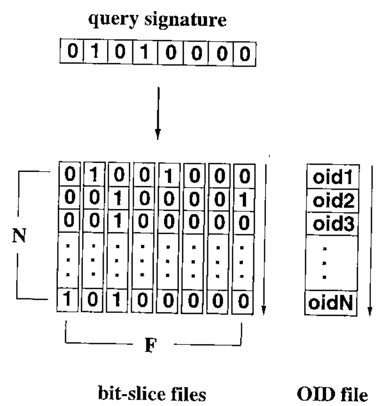

Fig. 3 File structure of BSSF.

3.2 Set Access Facilities for Nested Objects

Now we explain the file structures of four set access facilities for nested objects. $\mathcal{I}_{\text{BSSF}}$ is an extension of BSSF to facilitate set accesses to nested objects. $\mathcal{I}_{\text{NIX}}$ is based on the *nested index* (NIX) [1], [2], a B$^+$-tree-like indexing method proposed for nested objects. $\mathcal{I}_{\text{BSSF-NIX}}$ is a combination of the BSSF method and the NIX method, and $\mathcal{I}_{\text{NIX-NIX}}$ uses two NIX files.

An instance of a path $P = C_1.A_1.A_2.\cdots.A_n$ is expressed in the form $O_1.O_2.\ldots.O_n.\{v_1, v_2, \ldots, v_m\}$, where $\{v_1, v_2, \ldots, v_m\}$ is a set of primitive values or an OID set. When an instance of P, $O_1.O_2.\cdots.O_n.\{v_1, v_2, \ldots, v_m\}$ is given, the following entries are inserted in each facility.

$\mathcal{I}_{\text{BSSF}}$: The set signature S created from the set $\{v_1, v_2, \ldots, v_m\}$ is paired with O_1, the OID of the root object, and the pair $\langle S, O_1\rangle$ is stored in the BSSF file. If x instances of the path P are inserted, BSSF will have x entries.

$\mathcal{I}_{\text{NIX}}$: For each element of the set $\{v_1, v_2, \ldots, v_m\}$, the pair $\langle v_i, O_1\rangle$ $(1 \leqq i \leqq m)$ is created and inserted into the NIX file. Since the format of a leaf-node entry of NIX is ⟨key value, OID set⟩, if the pair $\langle v_i, O_1\rangle$ is inserted into NIX, the corresponding leaf-node entry becomes $\langle v_i, \{O_1, \ldots\}\rangle$.

$\mathcal{I}_{\text{BSSF-NIX}}$: The set signature S created from the set $\{v_1, v_2, \ldots, v_m\}$ is paired with O_n, the OID of the C_n object in the path P, and the pair $\langle S, O_n\rangle$ is inserted into the BSSF file. Next, the pair $\langle O_n, O_1\rangle$ is inserted into the NIX file.

$\mathcal{I}_{\text{NIX-NIX}}$: For each element of the set $\{v_1, v_2, \ldots, v_m\}$, the pair $\langle v_i, O_n\rangle$ $(1 \leqq i \leqq m)$ is created and inserted into an NIX file. This NIX file is called NIX_1. Next, the pair $\langle O_n, O_1\rangle$ is inserted into another NIX file. This NIX file is called NIX_2.

3.3 Query Processing Algorithms

In this subsection, query processing algorithms for the four set access facilities are described.

(1) $\mathcal{I}_{\text{BSSF}}$

Both $T \supseteqq Q$ and $T \subseteqq Q$ are processed as follows.

1. BSSF is searched based on the query condition and an OID set of C_1 objects is obtained. This set is called S_{OID}.
2. For each object in S_{OID}, a *forward traversal* [1], [2] is performed. When the forward traversal from $O_1 \in S_{\text{OID}}$ is performed, reachable C_n objects are retrieved. If at least one of the C_n objects satisfies the query condition ($T \supseteqq Q$, $T \subseteqq Q$), O_1 is included in the final result of this query and returned.

(2) $\mathcal{I}_{\text{NIX}}$

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 $T \supseteqq Q$, the intersection of the D_q sets is taken. For $T \subseteqq Q$, the union is taken.
3. For $T \supseteqq Q$, C_1 objects are retrieved based on the OID set and returned.

3′. For $T \subseteqq Q$, the following process is performed.

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

(3) $\mathcal{I}_{\text{BSSF-NIX}}$

Both $T \supseteqq Q$ and $T \subseteqq Q$ are processed as follows.

1. BSSF is searched based on the query condition and an OID set of C_n objects is obtained.
2. Each C_n object in the OID set is retrieved and checked as to 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.

(4) $\mathcal{I}_{\text{NIX-NIX}}$

1. For each element in the query set, NIX_1 is searched. Thus D_q OID sets of C_n objects are obtained.
2. For $T \supseteqq Q$, the intersection of the D_q sets is taken. For $T \subseteqq Q$, the union is taken.
3. Only for $T \subseteqq Q$, C_n objects are retrieved based on the OID set and checked as to whether they actually satisfy the query condition. Objects that do not satisfy the condition are removed from the OID set.
4. For each element of the OID set, NIX_2 is searched. Then an OID set of C_1 objects is obtained.
5. C_1 objects are retrieved based on the OID set.

3.4 Update Algorithms

Algorithms for inserting a new path instance $P = C_1.A_1.A_2.\cdots.A_n$ have already been described in Subsect. 3.2. Therefore, here we only show the deletion algorithms in the case that a C_n object O_n is deleted from the database and that *backward references* are not supported. Assume that O_n is already retrieved into memory and the A_n value of O_n is $\{v_1, v_2, \ldots, v_m\}$.

(1) $\mathcal{I}_{\text{BSSF}}$

1. A query signature is generated from $\{v_1, v_2, \ldots, v_m\}$ and BSSF is searched based on the set equality condition $(T \equiv Q)$ [11]. Namely, the target signatures which are the same as the query signature are searched. Thus an OID set of candidate C_1 objects is obtained.
2. For each object in the OID set, a forward traversal is performed. As a result, an OID set of C_1 objects that actually reference O_n are given. This set is called S_{OID}.
3. For each $O_1 \in S_{\text{OID}}$, the corresponding entry is deleted from the BSSF file.

(2) $\mathcal{I}_{\text{NIX}}$

Under the assumption that $A_1, \ldots, A_{n-1}$ may be set attributes, the deletion algorithm for NIX becomes complicated. A C_1 object O_1 may have multiple path instances to leaf-level objects. Therefore, O_1 has multiple set values corresponding to the multiple path instances, and they are not necessarily disjoint. Since NIX does not have counting information in its entries, we cannot delete $\langle v_i, O_1\rangle$ from the NIX leaf-node entry immediately. We may settle this problem by modifying the file structure of NIX, but here we present an algorithm based on the normal NIX file structure.

1. For each element v_i $(1 \leq i \leq m)$, NIX is searched. Thus m OID sets of C_1 objects are obtained. Next, the union is taken. This set is called S_{OID}.
2. For each object in S_{OID}, a forward traversal is performed and checked as to whether O_n is referenced. In this forward traversal, *all* reachable C_n objects are retrieved. As a result, an OID set of C_1 objects which actually reference O_n is obtained. Let this set be S'_{OID}.
3. For each object $O_1 \in S'_{\text{OID}}$, the referenced C_n objects are examined. If v_i $(1 \leq i \leq m)$ is not contained in the A_n value of any C_n objects other than O_n, the NIX entry $\langle v_i, O_1\rangle$ is deleted.

(3) $\mathcal{I}_{\text{BSSF-NIX}}$ and $\mathcal{I}_{\text{NIX-NIX}}$

Algorithms for $\mathcal{I}_{\text{BSSF-NIX}}$ and $\mathcal{I}_{\text{NIX-NIX}}$ are straightforward. For $\mathcal{I}_{\text{BSSF-NIX}}$, the entry of the BSSF file that corresponds to O_n is deleted and the entries in the NIX file that reference O_n are deleted. Deletions on $\mathcal{I}_{\text{NIX-NIX}}$ are processed in a similar manner.

4. Cost Models

In this section, cost models for the four set access facilities are developed. First, as a preliminary, cost formulas for BSSF and NIX are shown. Then, retrieval costs, storage costs, and update costs of the four set access facilities are derived. Only the number of page accesses will be taken into account as a cost factor. To simplify the estimation, we assume all target sets have an equal cardinality D_t.

In the cost models, the formula of Yao [13] is used to estimate the number of page accesses. To retrieve t records from n records stored on p pages, the number of page accesses is estimated by

$$npa(t, n, p) = p\left(1 - \prod_{i=1}^{t} \frac{n(1-1/p)-i+1}{n-i+1}\right).$$

4.1 Cost Formulas for BSSF and NIX

Here general cost formulas for BSSF and NIX are shown. They are revised versions of formulas in [8], [9]. Symbols and their definitions are summarized in Table 1.

Table 1 Symbols and their values.

Symbol	Definition and Value
D_t	cardinality of a target set
D_q	cardinality of a query set
N	number of objects
N_i	number of C_i objects
V	cardinality of the set domain of the attribute A_n $(= 10{,}000)$
P	size of a disk page $(= 4096$ bytes$)$
b	number of bits per byte $(= 8)$
oid	size of an OID $(= 8$ bytes$)$
P_o	number of page accesses to fetch an object $(= 1)$
$A\{c\}(N)$	total number of actual drops in N objects
F	signature size in bits
m	number of "1"'s (weight) in an element signature
$Fd\{c\}$	false drop probability
$SC_{bsf}(N)$	storage cost for a bit-slice file for N objects $(= \lceil \frac{N}{Pb} \rceil)$
$M\{c\}$	number of bit-slice files to be retrieved
$LC_{OID}\{c\}(N)$	access cost for the OID file for N objects
N_{oid}	number of OIDs in a disk page $(= \lfloor P/oid \rfloor = 512)$
$SC_{OID}(N)$	size of the OID file for N objects $(= \lceil N/N_{\text{oid}} \rceil$ pages$)$

(1) BSSF

The retrieval cost of BSSF, $RC_{\mathrm{BSSF}}\{c\}(N)$, is given in a general form:

$$\begin{aligned} &RC_{\mathrm{BSSF}}\{c\}(N) \\ &\quad = SC_{bsf}(N) \times M\{c\} + LC_{OID}\{c\}(N) \\ &\qquad + P_o(A\{c\}(N) + Fd\{c\}(N - A\{c\}(N))) \end{aligned} \tag{1}$$

$$\begin{aligned} &LC_{OID}\{c\}(N) \\ &\quad = npa(A\{c\}(N) \\ &\qquad + Fd\{c\}(N - A\{c\}(N)),\ N,\ SC_{OID}(N)), \end{aligned} \tag{2}$$

where c denotes the type of 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 is given as

$$M_{\{T \supseteq Q\}} = m_q \tag{3}$$

$$M_{\{T \subseteq Q\}} = F - m_q \tag{4}$$

[8], [11], where m_q is the expected number of "1"'s (*weight*) in the query signature and is given by $m_q \approx F(1 - e^{-\frac{m}{F}D_q})$.

The storage cost of BSSF is simply derived as

$$SC_{\mathrm{BSSF}}(N) = SC_{bsf}(N) \times F + SC_{OID}(N). \tag{5}$$

Cost formulas for updates, namely, IC_{BSSF} for insertion and $DC_{\mathrm{BSSF}}(N)$ for deletion, are derived as

$$IC_{\mathrm{BSSF}} = 2(m_t + 1) \tag{6}$$

$$DC_{\mathrm{BSSF}}(N) = \frac{SC_{OID}(N)}{2} + 2m_t + 1 \tag{7}$$

[9], where m_t is the weight of the target signature and is given by $m_t \approx F(1 - e^{-\frac{m}{F}D_t})$ [8].

(2) NIX

We derive cost formulas for NIX based on [3], [4]. Let x be the total number of keys and y be the number of entries corresponding to a key value in a leaf-node page of NIX. When z index keys are given, the retrieval cost of NIX is

$$\begin{aligned} &RC_{\mathrm{NIX}}(x, y, z) \\ &\quad = \begin{cases} \sum_{k=1}^{h} npa(t_k, n_k, p_k) & (z \geqq 2) \\ h & (z = 1) \end{cases} \end{aligned} \tag{8}$$

[4], where $t_h = z$ and $t_{k-1} = npa(t_k, n_k, p_k)$. n_k and p_k are the numbers of records and pages at level k of NIX and h is the height of NIX. These values are derived based on the parameters x and y using the cost formulas in [1].

The storage cost of NIX is

$$SC_{\mathrm{NIX}}(x, y) = \textstyle\sum_{k=1}^{h} p_k. \tag{9}$$

The insertion and the deletion costs based on z key values are

$$\begin{aligned} &IC_{\mathrm{NIX}}(x, y, z) = DC_{\mathrm{NIX}}(x, y, z) \\ &\quad = \begin{cases} \sum_{k=1}^{h} npa(t_k, n_k, p_k) \\ \qquad + npa(t_h, n_h, p_h) & (z \geqq 2) \\ h + 1 & (z = 1). \end{cases} \end{aligned} \tag{10}$$

The second term of this formula represents the rewrite cost.

(3) False Drop Probabilities and Actual Drops

False drop probability is an important measure for estimating the performance of signature files and is given by

$$Fd = \frac{\text{false drops}}{\text{total number of objects} - \text{actual drops}} \text{ [6].}$$

In this work, we use the following estimations [8], [11]:

$$Fd_{\{T \supseteq Q\}} \approx (1 - e^{-\frac{m}{F}D_t})^{mD_q} \tag{11}$$

$$Fd_{\{T \subseteq Q\}} \approx (1 - e^{-\frac{m}{F}D_q})^{mD_t}, \tag{12}$$

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]:

$$A_{\{T \supseteq Q\}}(N) = N \frac{{}_{V-D_q}C_{D_t - D_q}}{{}_{V}C_{D_t}} \tag{13}$$

$$A_{\{T \subseteq Q\}}(N) = N \frac{{}_{D_q}C_{D_t}}{{}_{V}C_{D_t}}. \tag{14}$$

4.2 Retrieval Costs

Before deriving retrieval cost formulas, we describe the parameters for nested objects. To simplify the derivation and analysis of costs, the following assumptions are made.

1. Each C_i object references fan_i^{i+1} C_{i+1} objects ($1 \leqq i \leqq n-1$).
2. No two objects share their references.

We call fan_i^{i+1} the *fanout* from C_i to C_{i+1}. For classes C_i and C_j ($i < j$), the fanout is defined as $fan_i^j = \prod_{k=i}^{j-1} fan_k^{k+1}$. Therefore, PI, the total number of instances of the path P, is given as $PI = N_1 fan_1^n$. The number of C_i objects is denoted by N_i.

(1) $\mathcal{I}_{\mathrm{BSSF}}$

Based on Eq. (1), the main part of the retrieval cost of $\mathcal{I}_{\mathrm{BSSF}}$ is given as

$$SC_{bsf}(PI) \times M\{c\} + LC_{OID}\{c\}(PI). \tag{15}$$

In the retrieval of $\mathcal{I}_{\mathrm{BSSF}}$, OIDs of C_1 objects are obtained rather than of C_n objects. Therefore, the third term of Eq. (1) must be modified. The number of OIDs of C_1 objects after duplicate elimination is estimated by Yao's formula:

$$\begin{aligned} &npa(A\{c\}(N_n) + Fd\{c\}(N_n - A\{c\}(N_n)), \\ &\quad N_n,\ N_n / fan_1^n). \end{aligned} \tag{16}$$

Then, forward traversals are performed for these C_1 objects. Therefore, the total retrieval cost is

$$RC\{\mathcal{I}_{\text{BSSF}},\, c\} = [\text{Eq. (15)}] + PFT([\text{Eq. (16)}],\, A_{\{c\}}(fan_1^n)), \tag{17}$$

where PFT is the forward traversal cost given in the Appendix.

(2) $\mathcal{I}_{\text{NIX}}$

For $T \supseteq Q$, the retrieval cost is derived as

$$RC\{\mathcal{I}_{\text{NIX}},\, T \supseteq Q\} = RC_{\text{NIX}}\left(V,\, \tfrac{D_t N_n}{V},\, D_q\right) + P_o A_{\{T \supseteq Q\}}(N_n). \tag{18}$$

For $T \subseteq Q$, the number of OIDs of C_1 objects obtained by the retrieval of NIX is given by

$$npa\left(N_n\left(1 - \tfrac{{}_{V-D_q}C_{D_t}}{{}_V C_{D_t}}\right),\, N_n,\, N_n/fan_1^n\right). \tag{19}$$

Therefore, the retrieval cost is

$$RC\{\mathcal{I}_{\text{NIX}},\, T \subseteq Q\} = RC_{\text{NIX}}\left(V,\, \tfrac{D_t N_n}{V},\, D_q\right) + PFT([\text{Eq. (19)}],\, A_{\{T \subseteq Q\}}(fan_1^n)). \tag{20}$$

(3) $\mathcal{I}_{\text{BSSF-NIX}}$

The retrieval cost of $\mathcal{I}_{\text{BSSF}}$ is

$$RC\{\mathcal{I}_{\text{BSSF-NIX}},\, c\} = RC_{\text{BSSF}\{c\}}(N_n) + RC_{\text{NIX}}(N_n,\, 1,\, A_{\{c\}}(N_n)) + P_o A_{\{c\}}(N_n). \tag{21}$$

The second term is the retrieval cost of the NIX file.

(4) $\mathcal{I}_{\text{NIX-NIX}}$

For $T \supseteq Q$, the retrieval cost is given as

$$RC\{\mathcal{I}_{\text{NIX-NIX}},\, T \supseteq Q\} = RC_{\text{NIX}}\left(V,\, \tfrac{D_t N_n}{V},\, D_q\right) + RC_{\text{NIX}}(N_n,\, 1,\, A_{\{T \supseteq Q\}}(N_n)) + P_o A_{\{T \supseteq Q\}}(N_n). \tag{22}$$

For $T \subseteq Q$, the retrieval cost is

$$RC\{\mathcal{I}_{\text{NIX-NIX}},\, T \subseteq Q\} = RC_{\text{NIX}}\left(V,\, \tfrac{D_t N_n}{V},\, D_q\right) + P_o N_n\left(1 - \tfrac{{}_{V-D_q}C_{D_t}}{{}_V C_{D_t}}\right) + RC_{\text{NIX}}(N_n,\, 1,\, A_{\{T \subseteq Q\}}(N_n)) + P_o A_{\{T \subseteq Q\}}(N_n). \tag{23}$$

The second term is the cost to check the C_n objects whose OIDs are retrieved in the first step (the retrieval of NIX_1).

4.3 Update Costs

The costs to insert a path instance $P = C_1.A_1.A_2.\cdots.A_n$ are given by

$$IC\{\mathcal{I}_{\text{BSSF}}\} = IC_{\text{BSSF}} \tag{24}$$

$$IC\{\mathcal{I}_{\text{NIX}}\} = IC_{\text{NIX}}\left(V,\, \tfrac{D_t N_n}{V},\, D_t\right) \tag{25}$$

$$IC\{\mathcal{I}_{\text{BSSF-NIX}}\} = IC_{\text{BSSF}} + IC_{\text{NIX}}(N_n,\, 1,\, 1) \tag{26}$$

$$IC\{\mathcal{I}_{\text{NIX-NIX}}\} = IC_{\text{NIX}}\left(V,\, \tfrac{D_t N_n}{V},\, D_t\right) + IC_{\text{NIX}}(N_n,\, 1,\, 1). \tag{27}$$

For $\mathcal{I}_{\text{BSSF}}$, the cost for deleting an object O_n is

$$DC\{\mathcal{I}_{\text{BSSF}}\} \approx SC_{bsf}(PI) \times nbs + npa(2,\, N_{\text{oid}} SC_{OID}(PI),\, SC_{OID}(PI)) + 2P_o n + 2 \tag{28}$$

[10], where nbs is the weight of a query signature that satisfies $Fd_{\{T \supseteq Q\}}(PI - A_{\{T \supseteq Q\}}(PI)) \approx 1$. The second term is the lookup cost of the OID file. The third and fourth terms represent the forward traversal cost and the rewrite cost, respectively.

The deletion cost for $\mathcal{I}_{\text{NIX}}$ is estimated as

$$DC\{\mathcal{I}_{\text{NIX}}\} = RC_{\text{NIX}}\left(V,\, \tfrac{D_t N_n}{V},\, D_t\right) + [\text{Eq. (30)}] + 2P_o D_t\left(1 - \tfrac{{}_{V-1}C_{D_t-1}}{{}_V C_{D_t}}\right)^{fan_1^n - 1} \tag{29}$$

$$FFT\left(npa\left(\left(1 - \tfrac{{}_{V-D_t}C_{D_t}}{{}_V C_{D_t}}\right) \times (N_n - 1) + 1,\, N_n,\, N_n/fan_1^n\right)\right) \tag{30}$$

[10], where Eq. (30) is the forward traversal cost. In this case, all referenced C_n objects must be traversed. Therefore the full forward traversal cost FFT (see Appendix) is used. The third term in Eq. (28) represents the read and rewrite costs for the NIX entries.

For $\mathcal{I}_{\text{BSSF-NIX}}$ and $\mathcal{I}_{\text{NIX-NIX}}$, the deletion costs are derived as

$$DC\{\mathcal{I}_{\text{BSSF-NIX}}\} = DC_{\text{BSSF}} + DC_{\text{NIX}}(N_n,\, 1,\, 1) \tag{31}$$

$$DC\{\mathcal{I}_{\text{NIX-NIX}}\} = DC_{\text{NIX}}\left(V,\, \tfrac{D_t N_n}{V},\, D_t\right) + DC_{\text{NIX}}(N_n,\, 1,\, 1). \tag{32}$$

4.4 Storage Costs

The storage costs for the four set access facilities as follows:

$$SC\{\mathcal{I}_{\text{BSSF}}\} = SC_{\text{BSSF}}(PI) \tag{33}$$

$$SC\{\mathcal{I}_{\text{NIX}}\} = SC_{\text{NIX}}\left(V,\, \tfrac{D_t N_n}{V}\right) \tag{34}$$

$$SC\{\mathcal{I}_{\text{BSSF-NIX}}\} = SC_{\text{BSSF}}(N_n) + SC_{\text{NIX}}(N_n,\, 1) \tag{35}$$

$$SC\{\mathcal{I}_{\text{NIX-NIX}}\} = SC_{\text{NIX}}\left(V,\, \tfrac{D_t N_n}{V}\right) + SC_{\text{NIX}}(N_n,\, 1). \tag{36}$$

5. Cost Analysis

Before comparing the retrieval costs of the four set access facilities, we describe the parameter settings. N_n, the number of objects in the class C_n, is set to 30,000. As the cardinality of A_n values, we consider two cases: $D_t = 10$ and $D_t = 100$. For the length of the path, we compare three cases of $n = 2, 3, 4$. The fanout parameters are set to $fan_i^{i+1} = fan$ $(1 \leq i \leq n-1)$. The constant fan value is set to 1, 5, or 10. For $\mathcal{I}_{\text{BSSF}}$ and $\mathcal{I}_{\text{BSSF-NIX}}$, it is necessary to set the BSSF parameters. We follow the following policy. 1) The storage costs of $\mathcal{I}_{\text{BSSF}}$ and $\mathcal{I}_{\text{BSSF-NIX}}$ are equal to or less than those of $\mathcal{I}_{\text{NIX}}$ and $\mathcal{I}_{\text{NIX-NIX}}$. This policy restricts the signature size F. When $D_t = 10$, we use $F = 500$ (bits) and when $D_t = 10$, $F = 5000$ (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 $T \supseteq Q$ are shown in Fig. 4 ($D_t = 10$) and Fig. 5 ($D_t = 100$). In this case, forward traversals are only performed by $\mathcal{I}_{\text{BSSF}}$. Therefore, the other three set access facilities do not depend on the fanout parameter fan or the path length n. The two figures show a similar tendency. Except for small D_q values (1 or 2), the retrieval costs are not different and increase monotonically. For small D_q values, $\mathcal{I}_{\text{BSSF}}$ configurations (especially $fan = 10$) give the worst costs. This is because $\mathcal{I}_{\text{BSSF}}$ needs forward traversals to process the query. In particular, when $fanout$ is large, more C_n objects correspond to one C_1 object so that the forward traversal cost increases. When $D_q = 1$, there are a considerable number of actual drops and false drops. Therefore, the overhead of the forward traversal cost determines the overall cost. However, when $D_q \geq 2$ or 3, drops are almost negligible, and the retrieval costs increase linearly. Although we have changed the path length n and examined the effect on the $\mathcal{I}_{\text{BSSF}}$ cost, it does not affect the retrieval costs very much.

The representative retrieval costs for $T \subseteq Q$ are shown in Fig. 6 ($D_t = 10$) and Fig. 7 ($D_t = 100$). In this query, $\mathcal{I}_{\text{BSSF}}$ and $\mathcal{I}_{\text{NIX}}$ need forward traversals. However,

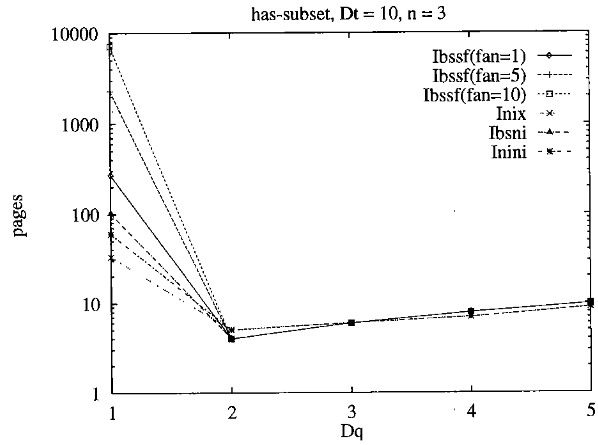

Fig. 4 Retrieval cost. ($T \supseteq Q$, $D_t = 10$, $n = 3$)

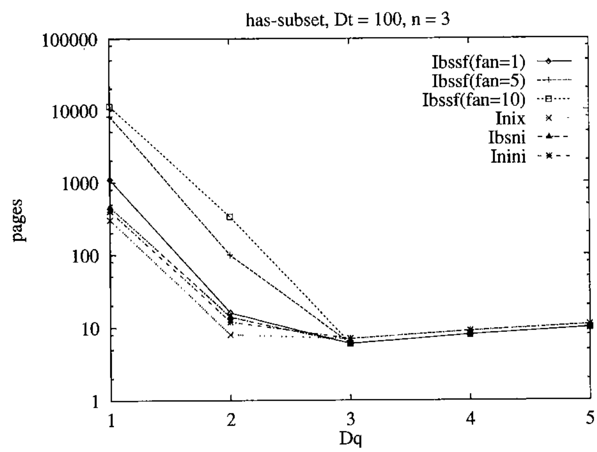

Fig. 5 Retrieval cost. ($T \supseteq Q$, $D_t = 100$, $n = 3$)

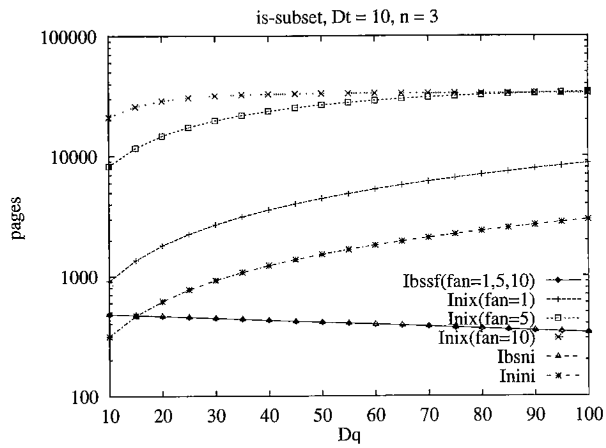

Fig. 6 Retrieval cost. ($T \subseteq Q$, $D_t = 10$, $n = 3$)

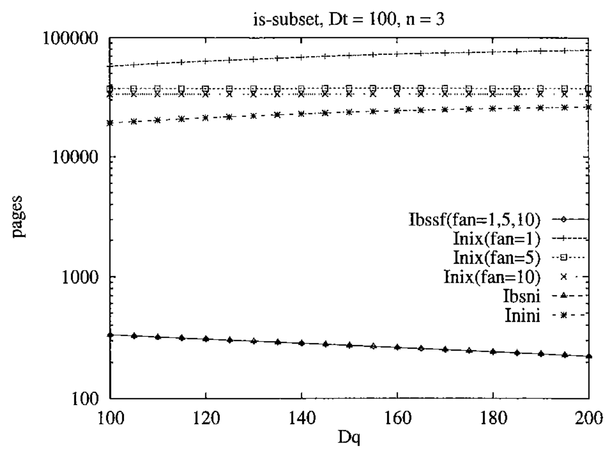

Fig. 7 Retrieval cost. ($T \subseteq Q$, $D_t = 100$, $n = 3$)

Table 2 Storage, insertion, and deletion costs.

	SC		IC		DC	
	D_t=10	D_t=100	D_t=10	D_t=100	D_t=10	D_t=100
$\mathcal{I}_{\rm BSSF}$	559	5059	21	197	12 †	12 †
$\mathcal{I}_{\rm NIX}$	629	10044	24	242	934 ‡	57600 ‡
					21200 §	33500 §
$\mathcal{I}_{\rm BSSF\text{-}NIX}$	662	5162	24	200	73	426
$\mathcal{I}_{\rm NIX\text{-}NIX}$	732	10147	27	245	27	245

† $n = 3$
‡ $n = 3$, $fan = 1$
§ $n = 3$, $fan = 10$

it seems that the retrieval cost of $\mathcal{I}_{\rm BSSF}$ does not suffer from the penalty of forward traversals and its cost is almost the same as that of $\mathcal{I}_{\rm BSSF\text{-}NIX}$. The reason is that the number of false drops of BSSF is very small for these D_q values so that few forward traversals occur.

When $D_t = 10$ and D_q is very small, the retrieval costs of $\mathcal{I}_{\rm BSSF}$ and $\mathcal{I}_{\rm BSSF\text{-}NIX}$ are higher than that of $\mathcal{I}_{\rm NIX\text{-}NIX}$. However, $\mathcal{I}_{\rm BSSF}$ and $\mathcal{I}_{\rm BSSF\text{-}NIX}$ are generally better in other cases. Furthermore, the retrieval costs of $\mathcal{I}_{\rm BSSF}$ and $\mathcal{I}_{\rm BSSF\text{-}NIX}$ can be improved by using the *smart retrieval strategy*, proposed in [8]. Therefore, for $T \subseteq Q$, $\mathcal{I}_{\rm BSSF}$ and $\mathcal{I}_{\rm BSSF\text{-}NIX}$ are considered to be superior to $\mathcal{I}_{\rm NIX}$ and $\mathcal{I}_{\rm NIX\text{-}NIX}$.

5.2 Storage, Insertion, and Deletion Costs

Storage, insertion, and update costs are shown in Table 2. When $D_t = 10$, the storage costs are almost the same. When $D_t = 100$, the storage costs of $\mathcal{I}_{\rm BSSF}$ and $\mathcal{I}_{\rm BSSF\text{-}NIX}$ are almost half those of $\mathcal{I}_{\rm NIX}$ and $\mathcal{I}_{\rm NIX\text{-}NIX}$. The four access facilities have almost the same insertion costs.

The deletion cost of $\mathcal{I}_{\rm BSSF}$ depends on n and that of $\mathcal{I}_{\rm NIX}$ depends on n and fan. Therefore, we show the representative costs for $\mathcal{I}_{\rm BSSF}$ and $\mathcal{I}_{\rm NIX}$. The cost of $\mathcal{I}_{\rm NIX}$ is prohibitively larger than those of the other access facilities. The reason is that $\mathcal{I}_{\rm NIX}$ needs many full forward traversals in deletion processing. The deletion cost of $\mathcal{I}_{\rm BSSF\text{-}NIX}$ is rather high, but we expect to reduce the cost by employing some techniques assumed in deriving Eq. (28) for $\mathcal{I}_{\rm BSSF}$ in [10].

6. Summary and Conclusions

In this paper, we have proposed four set access facilities, $\mathcal{I}_{\rm BSSF}$, $\mathcal{I}_{\rm NIX}$, $\mathcal{I}_{\rm BSSF\text{-}NIX}$, and $\mathcal{I}_{\rm NIX\text{-}NIX}$, for nested objects and compared their performance. We extended our cost models in [9] to more general situations in which nested objects may have set attributes in their nonleaf-level attributes. We developed revised algorithms and cost formulas, 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 shows that the four access facilities have similar performances except for small D_q values. When $D_q = 1$, $\mathcal{I}_{\rm BSSF}$ is the worst and $\mathcal{I}_{\rm NIX}$ is the best. However, for the retrieval cost for $T \subseteq Q$, $\mathcal{I}_{\rm BSSF}$ and $\mathcal{I}_{\rm BSSF\text{-}NIX}$ show relatively stable performance and are better than $\mathcal{I}_{\rm NIX}$ and $\mathcal{I}_{\rm NIX\text{-}NIX}$ for a reasonable range of D_q values. $\mathcal{I}_{\rm NIX}$ suffers performance degradation for $T \subseteq Q$ as fanout increases.

The storage costs of $\mathcal{I}_{\rm BSSF}$ and $\mathcal{I}_{\rm BSSF\text{-}NIX}$ are equal to or less than those of $\mathcal{I}_{\rm NIX}$ and $\mathcal{I}_{\rm NIX\text{-}NIX}$. All access facilities are almost equal as far as insertion costs are concerned. However, the deletion cost of $\mathcal{I}_{\rm NIX}$ is extremely high because of many full forward traversals. The deletion cost of $\mathcal{I}_{\rm BSSF\text{-}NIX}$ is slightly higher than those of $\mathcal{I}_{\rm BSSF}$ and $\mathcal{I}_{\rm NIX\text{-}NIX}$. However, there is room to improve the deletion cost of $\mathcal{I}_{\rm BSSF\text{-}NIX}$ by devising a smarter algorithm.

From our analysis, we can conclude that if we must select only one access facility from the four candidates, it is best to use $\mathcal{I}_{\rm BSSF\text{-}NIX}$ because of its stable performance and lower storage cost. The second-best candidate is $\mathcal{I}_{\rm BSSF}$. If the case of $T \supseteq Q$ and $D_q = 1$ is important, $\mathcal{I}_{\rm NIX}$ may be another candidate. However, it cannot support $T \subseteq Q$ queries very well.

Further study of set access facilities for nested objects is ongoing in our group. The research issues include processing of another type of set query (e.g., set equality) and application of the smart retrieval strategy [8].

Acknowledgement

The authors are grateful to members of Laboratory for Database and Media Systems (Uemura Laboratoty), NAIST, and Database Laboratory, Univ. of Tsukuba, for their encouragement and support.

References

[1] E. Bertino and W. Kim, "Indexing techniques for queries on nested objects," IEEE Trans. Knowl. and Data Eng., vol.1, no.2, pp.196–214, June 1989.

[2] E. Bertino, "A survey of indexing techniques for object-oriented database management systems," in Query Processing for Advanced Database Systems, eds. J.C. Freytag, D. Maier and D. Vossen, Chapter 13, Morgan Kaufmann, San Mateo, CA, 1993.

[3] E. Bertino, "Index configuration in object-oriented databases," VLDB Journal, vol.3, no.3, pp.355–399, July 1994.
[4] S. Choenni, E. Bertino, H.M. Blanken, and T. Chang, "On the selection of optimal index configuration in OO databases," Proc. Int. Conf. on Data Engineering, Houston, Texas, pp.526–537, Feb. 1994.
[5] C. Faloutsos and S. Christodoulakis, "Signature files: an access method for documents and its analytical performance evaluation," ACM Trans. Office Infor. Syst., vol.2, no.4, pp.267–288, Oct. 1984.
[6] C. Faloutsos and R. Chen, "Fast text access methods for optical and large magnetic disks: designs and performance comparison," Proc. Int. Conf. on VLDB, Los Angeles, CA, pp.280–293, Aug. 1988.
[7] C. Faloutsos, "Signature-based text retrieval methods: a survey," IEEE Database Eng., vol.13, no.1, pp.25–32, March 1990.
[8] Y. Ishikawa, H. Kitagawa, and N. Ohbo, "Evaluation of signature files as set access facilities in OODBs," Proc. ACM SIGMOD Conf., Washington, D.C., pp.247–256, May 1993.
[9] Y. Ishikawa and H. Kitagawa, "Analysis of indexing schemes to support set retrieval of nested objects," Proc. Int. Symp. on Advanced Database Technologies and Their Integration (ADTI'94), Nara, Japan, pp.55–62, Oct. 1994.
[10] Y. Ishikawa and H. Kitagawa, "Design and performance analysis of indexing schemes for set retrieval of nested objects," Technical Report, NAIST-IS-TR95018, Nara Institute of Science and Technology, May 1995.
[11] H. Kitagawa, Y. Fukushima, Y. Ishikawa, and N. Ohbo, "Estimation of false drops in set-valued object retrieval with signature files," Proc. of 4th Int. Conf. on Foundations of Data Organization and Algorithms (FODO), pp.146–163, Oct. 1993 (LNCS 730).
[12] R. Sacks-Davis, A. Kent, and K. Ramamohanarao, "Multikey access methods based on superimposed coding techniques," ACM Trans. Database Syst., vol.12, no.4, pp.655–696, Dec. 1987.
[13] S.B. Yao, "Approximating block accesses in database organizations," Commun. ACM, vol.20, no.4, pp.260–261, April 1977.

Appendix: Forward Traversal Costs

Suppose that x C_1 objects are given. We derive the expected number of page accesses in forward traversals from the C_1 objects to the descendant C_n objects. We assume the use of the *nested-loop forward traversal* method [2] and the assumptions made in Sect. 4.

There exist two cases for forward traversals: 1) the traversal cannot be finished until all reachable C_n objects are obtained (*full forward traversal*); 2) the traversal can be finished at the time a C_n object satisfying the condition is found (*partial forward traversal*). The full forward traversal cost FFT is derived as

$$FFT(x) = P_o x \sum_{i=1}^{n} fan_1^i. \tag{A·1}$$

The partial forward traversal cost PFT is given as

$$PFT(x, y) = P_o x \sum_{i=1}^{n} e_i(y) \tag{A·2}$$

$$e_i(y) = \lceil e_{i+1}(y)/fan_i^{i+1} \rceil \quad (1 \leqq i \leqq n-1) \tag{A·3}$$

$$e_n(y) = e(fan_1^n, y), \tag{A·4}$$

where y ($y \geqq 1$) is the number of C_n objects satisfying the condition per C_1 object. The function $e(p, q)$ is

$$e(p, q) = \frac{q}{p} + \sum_{i=2}^{p-q+1} \left(\frac{iq}{p-i+1} \times \prod_{j=0}^{i-2} \left(1 - \frac{q}{p-j}\right)\right). \tag{A·5}$$

When $y < 1$, PFT is

$$PFT(x, y) = PFT(xy, 1) + FFT(x(1-y)). \tag{A·6}$$

Details of deriving Eqs. (A·2)–(A·6) are given in [10].

Yoshiharu Ishikawa received the B.S., M.E., Dr.Eng. degrees in information engineering from University of Tsukuba in 1989, 1991 and 1995, respectively. He is currently a research associate at Graduate School of Information Science, Nara Institute of Science and Technology, Nara, Japan. His research interests include the design and implementation of advanced DBMSs and scientific databases. He is a member of the ACM, IEEE Computer Society, IPSJ, and JSSST.

Hiroyuki Kitagawa received the B.Sc. degree in physics and the M.Sc. and D.Sc. degrees in computer science, all from the University of Tokyo, in 1978, 1980, and 1987, respectively. From 1981 to 1988, he was with the Software Product Engineering Laboratory of NEC Corporation. In 1988, he joined the Institute of Information Sciences and Electronics, University of Tsukuba, where he is currently an Associate Professor. His research interests include database system architecture, object database systems, temporal database and version management, text databases, and heterogeneous information resource management. Dr. Kitagawa is a member of ACM, IEEE Computer Society, IPSJ, and JSSST.