# シグネチャファイルによる集合値検索のコスト評価

石川佳治† 北川博之†† 大保信夫††

集合データは基本的なデータ構造であり，複合オブジェクトの部分構造としても頻繁に現れる．このため，集合値に関する検索条件を効率的に支援する索引機構は，先進的な応用分野を対象としたデータベースシステムにおいて重要なものとなる．筆者らは，テキスト検索で従来用いられてきたシグネチャファイルの手法を集合値検索に適用することを提案し，比較的小規模のデータベースを想定し，コスト評価や効率的な問い合わせ処理方式などの検討を行ってきた．本論文では，中規模データベースに対するシグネチャファイルの有用性の評価を行う．シグネチャファイルの構成手法としては，ビットスライストシグネチャファイル (bit-sliced signature file, BSSF) を対象とし，入れ子型インデックス (nested index) を比較の対象とする．中規模データベースにおける検索コスト，記憶コスト，更新コストの評価を行い，小規模データベースにおける評価結果と比較する．また，中規模データベースにおけるシグネチャファイルの性能向上のためには，ビットスライストシグネチャファイルに圧縮を用いることが有効であると考え，ファイル圧縮時のコストについても評価を行い，その有効性を示す．

# Cost Evaluation of Set-valued Object Retrieval with Signature Files

Yoshiharu Ishikawa,† Hiroyuki Kitagawa†† and Nobuo Ohbo††

Set-valued data is a primitive data object and often appears as a sub-structure in complex objects. Therefore, access facilities which can support set-valued object retrieval become important for databases supporting advanced application areas. We have proposed the use of signature files for efficient set-valued object retrieval, and studied their performance and efficient query processing strategies for relatively small databases. In this paper, we evaluate the availability of signature files applied to medium-scale databases. As a signature file organization, we use the bit-sliced signature file (BSSF) and compare its performance with that of the nested index. We evaluate retrieval costs, storage costs, and update costs of signature files applied to medium-scale databases, and compare the result with that for small-scale databases. Furthermore, to improve the performance of signature files in medium-scale databases, we evaluate the costs of the compressed BSSF and discuss the effect of compression. We show that the compressed BSSF is promising for set-valued object retrieval in medium-scale databases.

## 1. はじめに

集合は基本的なデータ構造の一つであり，複合オブジェクト[10]の部分構造としても頻繁に現れる．また，複合オブジェクトに対する問い合わせの一部には，直接的には集合値に関する検索条件を含まないが，問い合わせ処理において集合値に関する検索処理へと帰着可能なものが存在する[8]．このような理由により，集合値に対する索引機構は，集合値を直接的に扱うデータベースのみでなく，複合オブジェクトを対象とするデータベースにおいても重要なものとなる．複合オブジェクトに対する索引機構としては，これまで入れ子型インデックス (nested index)，パスインデックス (path index)，マルチインデックス (multiindex) などが提案され，性能の解析も行われてきた[1]．しかし，これらの索引機構は複合オブジェクトがその一部としてもつ単純属性値に基づく検索の支援を指向したものであり，集合値に関するさまざまな検索処理を考慮したものではない．

このような背景に基づき，筆者らは集合値に関する検索を効率的に支援する索引機構の研究を進めてきた[6),8),9),11]．本論文で述べる索引機構は，テキスト

† 奈良先端科学技術大学院大学情報科学研究科
Graduate Institute of Information Science, Nara Institute of Science and Technology
†† 筑波大学電子・情報工学系
Institute of Infomation Sciences and Electronics, University of Tsukuba

データに対するキーワード検索などで用いられてきた，シグネチャファイル[2),5)]の手法をもとにしたものである．

文献 8) では，シグネチャファイルの物理的な構成手法の候補としてシーケンシャルシグネチャファイル(SSF)とビットスライストシグネチャファイル(BSSF)を取り上げ，比較的小規模のデータベース（総オブジェクト数 $N=32,000$）を想定してそれらの検索コスト，記憶コスト，更新コストを入れ子型インデックスと比較した．検索の種類としては，包含関係($\supseteq$)に関する2種類の問い合わせ（$T\supseteq Q, T\subseteq Q$）を対象とした*．その結果，SSF は他の二つの索引機構に比べ検索コストにおいて大きく劣ることが明らかとなった．また，検索コストの解析により，BSSF と入れ子型インデックスではさらに効率的な検索処理（スマート検索）が可能であることを示し，改良された問い合わせ処理方式に基づく両者の検索コストを比較した．$T\supseteq Q$ の問い合わせについては BSSF と入れ型インデックスの検索コストはほぼ同等となり，$T\subseteq Q$ の問い合わせについては BSSF が優れていることが明らかとなった．記憶コストにおいても BSSF は入れ子型インデックスに対して有利であり，この点からも BSSF 方式に基づくシグネチャファイルが集合値索引機構として有望であることを示した．

本論文では，データベースの規模が拡大された場合での，集合値索引機構としてのシグネチャファイルの有用性について評価を行う．中規模データベースを想定し，総オブジェクト数 $N=320,000$ の場合のコスト評価を行い，小規模データベースを想定した $N=32,000$ の場合のコスト評価結果と比較する．文献 8) における評価に基づき，シグネチャファイルの物理的な構成手法としては検索コストで有利である BSSF 方式を対象とする．比較の対象としては入れ子型インデックスを用い，検索コスト，記憶コスト，更新コストについて比較を行う．検索コストに関しては，スマート検索を用いた場合の比較を中心に行う．中規模データベースに関するコスト評価を可能とするため，入れ子型インデックスについては文献 8) で用いたコストモデルの拡張を行う．

また，本論文では，ファイルの圧縮により BSSF の検索・記憶コストの向上をはかることを提案する．圧縮を用いた BSSF に対するコスト評価を行い，中規模データベースにおける集合検索機構として有望であることを示す．

本論文の構成は以下のとおりである．2章では，集合値検索の概念とシグネチャファイルの手法の集合値検索への応用について述べる．3章では，性能評価のためのコストモデルについて述べる．4章では，検索コスト，記憶コスト，更新コストの面から，BSSF と入れ子型インデックスについてのコスト評価を行う．検索コストに関しては，コストの傾向を検討することにより，より効果的な検索方式であるスマート検索を導き，これを基にコストの比較を行う．中規模データベースに関する評価結果を小規模データベースの場合と比較し，さらに BSSF についてはファイルの圧縮を行った場合のコストについても評価する．5章では本論文のまとめを行う．

## 2. シグネチャファイルを用いた集合値検索

本章では，集合値検索の概念，および，集合値索引機構としてのシグネチャファイルの概念について簡単に説明する．

### 2.1 集合値検索について

例として学生に関するデータベースを考える．Student クラスには属性として name と hobbies があり，hobbies 属性は文字列の集合を値とする．ここで，以下の問い合わせ $Q_1$ を考える．

$Q_1$ : **select** name
**from** Student
**where** hobbies$\supseteq$ {"Baseball", "Fishing"}

$Q_1$ は，"Baseball" と "Fishing" の両方を趣味（の一部）とするような学生の名前を求める問い合わせである．問い合わせ条件中に現れる集合 {"Baseball", "Fishing"} を**問い合わせ集合** (*query set*, $Q$) と呼び，データベース中に格納され問い合わせ集合との比較の対象となる，それぞれの集合値を**ターゲット集合** (*target set*, $T$) と呼ぶ．$Q_1$ は，問い合わせ集合を部分集合として含むターゲット集合に対応するデータオブジェクトを検索する問い合わせである．このような問い合わせを，$T\supseteq Q$ (*has-subset*) の問い合わせと呼ぶ．これに対し，問い合わせ集合の部分集合となるターゲット集合に対応するデータオブジェクトを検索する問い合わせを $T\subseteq Q$ (*is-subset*) の問い合わせと呼ぶ．$T\supseteq Q$ の特殊な場合として，問い合わせ集合の要素数が1である場合の $T\ni q$ (*has-element*) が存在

* $T$ はデータベース中の集合，$Q$ は問い合わせ条件で与えられる集合を表している．2 種類の問い合わせの詳細については後述する．

するが，これについては $T \supseteq Q$ に含めて議論する．

### 2.2 集合値索引機構としてのシグネチャファイル

**シグネチャファイル** (*signature file*)[2),5)] は，主にテキストデータベースにおけるキーワード検索において利用されてきた索引手法である．**シグネチャ** (*signature*) とは，個々のデータオブジェクトから生成される固定長のビット列であり，シグネチャを対応するデータオブジェクトの識別子 (OID) とともに格納したものがシグネチャファイルである．シグネチャの生成には，**スーパーインポーズドコーディング** (*superimposed coding*) 呼ばれるコーディング手法[3)]が用いられるのが一般的である．以下で述べる集合シグネチャの生成方式もスーパーインポーズドコーディングに基づいている．**表1**に，本論文で用いる，シグネチャファイルに関する記号を示す．

集合シグネチャは以下のように作成される．集合値が与えられると，まず，ハッシュ法などにより，集合の各要素から**要素シグネチャ** (*element signature*) が作られる．要素シグネチャは $F$ ビットのビット列であり，そのうちの $m$ ビットに“1”，残りの $F-m$ ビットに“0”が設定される．$m$ を要素シグネチャの**ウェイト** (*weight*) と呼ぶ．次に，すべての要素シグネチャのビットごとの論理和をとることにより，**集合シグネシャ** (*set signature*) が作成される．集合シグネチャにおける“1”の数を集合シグネチャのウェイトと呼ぶ．**図1**に，集合 {“Baseball”, “Golf”, “Fishing”} に対して集合シグネチャを生成する例を示す．作成された集合シグネチャは，対応する OID とともにシグネチャファイルに格納される．シグネチャファイルに格納されるそれぞれの集合シグネチャを**ターゲットシグネチャ** (*target signature*) と呼ぶ．

問い合わせが与えられると，まず，問い合わせ集合より**問い合わせシグネチャ** (*query signature*) と呼ばれる集合シグネチャが作成される．次に，シグネチャファイル中のそれぞれのターゲットシグネチャについて，問い合わせシグネチャとの間に問い合わせの種類によって定まるマッチング条件が成り立っているかどうかが調べられる．マッチング条件が満たされた場合には，対応するデータオブジェクトは問い合わせを満たす候補となる．このようなデータオブジェクトは**ドロップ** (*drop*) と呼ばれる．それぞれの問い合わせに対するマッチング条件は以下のようになる．

$T \supseteq Q$ : 〈問い合わせシグネチャ〉∧〈ターゲットシグネチャ〉≡〈問い合わせシグネチャ〉

$T \subseteq Q$ : 〈問い合わせシグネチャ〉∧〈ターゲットシグネチャ〉≡〈ターゲットシグネチャ〉

ここで，’∧’ はビット列の論理積を，’≡’ はビット列が等しいことを示している．

問い合わせ処理の最後にそれぞれのドロップが検索され，それらが実際に問い合わせ条件を満たすかどうかが調べられる．この処理は**フォルスドロップレゾリューション** (*false drop resolution*) と呼ばれる．フォルスドロップレゾリューションの結果，条件を満たしたデータオブジェクトは**アクチュアルドロップ** (*actual drop*) と呼ばれ，条件を満たさなかったものは**フォルスドロップ** (*false drop*) と呼ばれる．

**表 1** シグネチャファイルに関する記号
Table 1 Symbols for signature file.

| 記　号 | 定　　義 |
|---|---|
| $F_d$ | フォルスドロップ確率 |
| $F$ | シグネチャのビット長 |
| $m$ | 要素シグネチャのウェイト |
| $m_t$ | ターゲットシグネチャのウェイトの期待値 |
| $m_q$ | 問い合わせシグネチャのウェイトの期待値 |
| $D_t$ | ターゲット集合の要素数 |
| $D_q$ | 問い合わせ集合の要素数 |

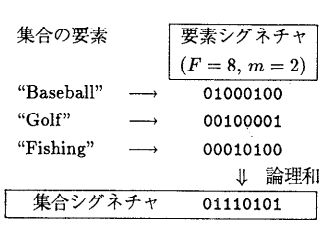



**図 1** 集合シグネチャの生成
Fig. 1 Creation of a set signature.

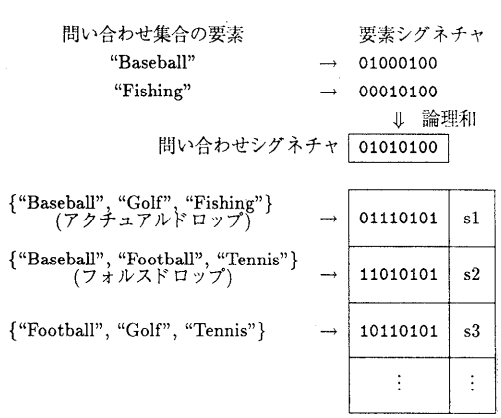



**図 2** 問い合わせ処理 ($T \supseteq Q$)
Fig. 2 Query processing ($T \supseteq Q$).

先に示した問い合わせ $Q_1(T \supseteq Q)$ に対する問い合わせ処理の例を**図2**に示す．

### 2.3 フォルスドロップ確率

**フォルスドロップ確率**(*false drop probability*) $F_d$ は，シグネチャファイルの性能を評価するための重要な尺度であり，以下の式で与えられる[4]．

$$F_d = \frac{\text{フォルスドロップ数}}{N - \text{アクチュアルドロップ数}}$$

ここで，$N$ はシグネチャファイルに格納されるエントリの総数である．

要素シグネチャにおいて“1”の値をもつビットが一様に分布し，要素シグネチャのウェイト $m$ がシグネチャのビット長 $P$ に比べ十分に小さい ($m \ll F$) という条件のもとで，$T \supseteq Q$, $T \subseteq Q$ に対するフォルスドロップ確率は以下の近似式で与えられる[8),11)]．

$$F_{d\{T \supseteq Q\}} \approx (1 - e^{-\frac{m}{F} D_t})^{m D_q} \quad (1)$$

$$F_{d\{T \subseteq Q\}} \approx (1 - e^{-\frac{m}{F} D_q})^{m D_t} \quad (2)$$

ここで，$D_q$ は問い合わせ集合の要素数，$D_t$ はターゲット集合の要素数である．文献11)では，これらの近似式に基づくフォルスドロップ確率の見積りの妥当性をシミュレーションにより検証した．

## 3. コストモデルについて

本論文では，特に中規模のデータベースを対象として，集合値検索機構としてのシグネチャファイルの性能の評価を行う．シグネチャファイルの物理的な構成手法についてはさまざまな提案がなされているが，本論文ではその一つである**ビットスライストシグネチャファイル**(*Bit-Sliced Signature File*, *BSSF*) を評価の対象とする．**図3**に BSSF の構成を示す．BSSF ではシグネチャはビットごとに別々のファイルに格納され，結果として $F$ 個のファイルが作成される．これらのファイルは**ビットスライスファイル**，または簡単に**ビットスライス**と呼ばれる．検索時には $F$ 個のビットスライスのうちの一部にのみアクセスすればよいため，検索コストの面で有利となる．また，本論文では，中規模のデータベースを考慮し，新たにファイル圧縮を用いた BSSF についても性能の評価を行う．このため，文献8) で用いた BSSF のコストモデルに拡張を行う．

性能比較の対象としては，複合オブジェクトに対する索引機構として提案されている入れ子型インデックス (nested index)[1)]を用いる．入れ子型インデックスについても，中規模データベースを考慮して，文献8) で示したコストモデルの拡張を行う．

索引機構の検索コストは主に入出力に依存するため，本論文ではページアクセス数により検索コストを評価する．**表2**にはコスト式において用いられる定数

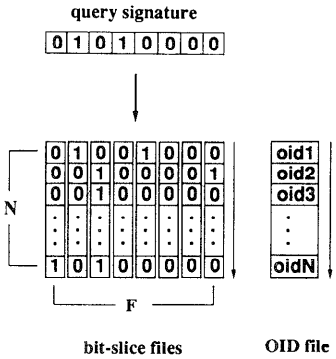



図3 BSSF の構成
Fig. 3 File structure of BSSF.

表2 定数記号
Table 2 Constant symbols.

| 記号 | 定義および値 |
|---|---|
| $N$ | オブジェクトの総数 (=32,000 or 320,000) |
| $P$ | 1ディスクページのバイト数 (=4,096) |
| $oid$ | OID のバイト数 (=8) |
| $V$ | 集合要素の定義域の要素数 (=13,000) |
| $b$ | 1バイトのビット数 (=8) |
| $O_n$ | 1ディスクページ中に格納できる OID の数 ($O_n = \lfloor P/oid \rfloor = 512$) |
| $SC_{OID}$ | OID ファイルのページ数 ($=\lceil N/O_n \rceil$) |
| $P_s$, $P_u$ | 1オブジェクトへのアクセスコスト ($P_s$：検索成功の場合，$P_u$：検索失敗の場合) ($P_s = P_u = 1$ ページ) |

表3 パフォーマンスの尺度となる記号
Table 3 Performance measures.

| 記号 | 定義 |
|---|---|
| $RC$ | 検索コスト (単位：ページ) |
| $LC_{OID}$ | OID ファイルへのアクセスコスト (単位：ページ) |
| $SC$ | 記憶コスト (単位：ページ) |
| $SC_{bsf}$ | 一つのビットスライスファイルの記憶コスト (単位：ページ) |
| $M$ | 検索対象のビットスライス数 |
| $A$ | アクチュアルドロップ数 |
| $\alpha$ | OID ファイル1ページあたりのアクチュアルドロップ数 |
| $UC_I$ | 集合値の挿入コスト (単位：ページ) |
| $UC_D$ | 集合値の削除コスト (単位：ページ) |

記号を示す．また，**表3**にはパフォーマンスを評価する尺度となる記号を示す．コスト式を導く上で以下の仮定を行う[8]．

1. ターゲット集合の要素数は，データベース中のすべてのオブジェクトについて一様に定数値 $D_t$ である．4章では $D_t=10$ および $D_t=100$ の場合について解析を行う．
2. 集合要素の定義域（要素数 $V$）の $D_t$ 個の要素からなる集合が，いずれも等確率でターゲット集合として出現する．
3. OID による1データオブジェクトのアクセスコストは1ページである．

### 3.1 BSSF のコスト式

本節では，BSSF の検索コスト，記憶コスト，更新コストのコスト式を求める．2.2 節では，集合値索引機構としてのシグネチャファイルの問い合わせ処理を一般的な形で述べたが，BSSF の検索コストを考える上では，ファイル構成に基づくより具体的な問い合わせ処理を考える必要がある．BSSF による，$T\supseteq Q$ に対する問い合わせ処理のステップは以下のようになる．

1. 問い合わせ条件より問い合わせシグネチャを作成する．
2. 問い合わせシグネチャにおいて“1”の値をもつ各ビット位置に対応するビットスライスが検索される．検索されるビットスライス数の期待値は，問い合わせシグネチャのウェイトの期待値 $m_q$ であり，以下の式で与えられる[8]．

$$m_q\approx F(1-e^{-\frac{m}{F}D_q}) \tag{3}$$

3. ステップ2で読み出したすべてのビットスライスの論理積をとる．
4. 論理積の結果において“1”の値をもつエントリについて，対応する OID を OID ファイルから検索する．この OID をもとにデータオブジェクトを検索し，フォルスドロップレゾリューションを行う．最後に，問い合わせ条件を満たしたデータオブジェクトが問い合わせ結果として返される．

$T\subseteq Q$ に対する問い合わせ処理は，$T\supseteq Q$ の処理で“1”と“0”の役割を入れ換えたものである[8]．検索されるビットスライス数の期待値は $F-m_q$ となる．

**検索コスト**

BSSF の検索コスト $RC$ は以下のように表すことができる．

$$RC=SC_{bsf}M+LC_{OID}+P_sA+P_uF_d(N-A) \tag{4}$$

$$LC_{OID}=SC_{OID}\times\min(F_d(O_u-\alpha)+\alpha,1) \tag{5}$$

(4)式の第一項はビットスライスの検索に要するコストを表している．$SC_{bsf}$ は一つのビットスライスの記憶コストであり，ファイルの圧縮を考えない場合，$SC_{bsf}=\left\lceil\frac{N}{Pb}\right\rceil$ である．$M$ は検索されるビットスライス数の期待値であり，上の議論より，

$$M_{\{T\supseteq Q\}}=m_q \tag{6}$$

$$M_{\{T\subseteq Q\}}=F-m_q \tag{7}$$

となる．$LC_{OID}$ は OID ファイルの検索に要するコストである．$A$ はアクチュアルドロップ数の期待値であり，

$$A_{\{T\supseteq Q\}}=N\frac{V-D_q C_{D_t-D_q}}{V C_{D_t}} \tag{8}$$

$$A_{\{T\subseteq Q\}}=N\frac{D_q C_{D_t}}{V C_{D_t}} \tag{9}$$

で与えられる[8]．$\alpha$ は OID ファイルの1ページ当たりのアクチュアルドロップ数の期待値であり，

$$\alpha=\frac{A}{SC_{OID}} \tag{10}$$

である．$P_sA+P_uF_d(N-A)$ はフォルスドロップレゾリューションに要するコストであり，$P_sA$ がアクチュアルドロップに関するコストを，$P_uF_d(N-A)$ がフォルスドロップに関するコストをそれぞれ表している．

**圧縮時のビットスライスのサイズ**

後述する BSSF のパラメータ設定のもとでは，ビットスライスの多くのビットの値が“0”となる．したがって，ファイル圧縮によるサイズの減少の度合が大きいと考えられ，検索コスト・記憶コストの向上が期待できる．以下では，圧縮を行った場合のビットスライスのビット長を予測する．圧縮の手法としてはランレングス符号化を用いる．

$\theta$ を“0”が生じる確率とし，$l$ を

$$\theta^l+\theta^{l+1}\leq 1<\theta^l+\theta^{l-1}$$

を満たす整数としたとき，最適なコード化を行った場合のコードのビット長の期待値 $\bar{n}$ は以下のように与えられる[7]．

$$\bar{n}=\lceil\log_2 l\rceil+1+\frac{\theta^k}{1-\theta^l} \tag{11}$$

ただし，$k=2^{\lfloor\log_2 l\rfloor+1}-l$ である．

BSSF に格納されるシグネチャ（ターゲットシグネチャ）のあるビットの値が“1”となる確率が

$$p(b_t)=1-\left(1-\frac{m}{F}\right)^{D_t}\approx 1-e^{-\frac{mD_t}{F}}$$

で与えられることにより[11]，$\theta$ は

$$\theta=1-p(b_t)=e^{-\frac{mD_t}{F}} \quad (12)$$

となる．以上をもとに，圧縮後の1ビットスライスのページサイズの期待値は，

$$SC_{bsf}=\left\lceil\frac{Np(b_t)\bar{n}}{Pb}\right\rceil \quad (13)$$

で与えられる．$Np(b_t)$ は1ビットスライス中の“1”の値を持つビット数の期待値，すなわち，“0”のビットのシーケンス数の期待値である．(13)式を用いてファイル圧縮時の検索コストが求められる．

**記憶コスト**

BSSF の記憶コストは，$F$ 個のビットスライスファイルと OID ファイルの記憶コストの合計として，以下のように与えられる．

$$SC=SC_{bsf}F+SC_{OID} \quad (14)$$

**更新コスト**☆

BSSF の更新コストを導く上で，ファイル作成時にビットスライスファイルのすべてのビットが“0”に初期化されるという仮定をおく．また，シグネチャの挿入では，シグネチャはファイルの末尾に追加されるものとする．挿入時には，挿入されるシグネチャで“1”が立っているビット位置に対応するビットスライスファイルと，OID ファイルの更新が必要となる．ターゲットシグネチャにおけるウェイトの期待値 $m_t$ が，

$$m_t=F(1-e^{-\frac{m}{F}D_t}) \quad (15)$$

で与えられることより[8]，挿入コストは

$$UC_I=m_t+1 \quad (16)$$

となる．

シグネチャの削除の場合には，削除対象のシグネチャ自身とその対応する OID が与えられるものとする．削除時には，まず OID ファイルに削除フラグが立てられる．これには $\frac{SC_{OID}}{2}$ ページのアクセスが必要となる．次に，削除対象のシグネチャにより“1”が立てられていたビットスライスのビットの値が“0”に戻される．よって，削除コスト $UC_D$ は

$$UC_D=m_t+\frac{SC_{OID}}{2} \quad (17)$$

となる．

### 3.2 入れ子型インデックスのコスト式

入れ子型インデックスは B-木に基づく索引手法であり，リーフページには，キー値とそのキー値に対応する OID のリストからなるインデックスエントリが格納される．ノンリーフノードの形式は B-木と同様である．文献 1) では，ネスト構造をもった複合オブジェクトに対する索引機構という立場から，入れ子型インデックス，パスインデックス，およびマルチインデックスのコストの比較が行われている．本節では，文献 1) のコストモデルに対し集合値検索の立場から拡張を行ったコストモデル[8]を，中規模のデータベースを考慮してさらに拡張する．なお，以下では入れ子型インデックスを NIX と略する．**表4**に NIX に関する記号を示す．他の記号については BSSF と共通である．

**表 4** NIX に関する記号
Table 4 Symbols for NIX.

| 記号 | 定義および値 |
|---|---|
| $ll$ | リーフノードエントリのバイト数 |
| $d$ | 与えられた要素値をその集合属性値の中に含むようなオブジェクトの平均数 |
| $kl$ | キー値を格納するフィールドのバイト数（＝8バイト） |
| $noid$ | OID エントリの個数を格納するフィールドのバイト数（＝2バイト） |
| $lp$ | リーフページの総数 |
| $nlp$ | ノンリーフページの総数 |
| $f$ | ノンリーフノードの平均のファンアウト（=218） |
| $rc$ | 1インデックスエントリの検索のためのページアクセス数 |
| $ds$ | ディレクトリのバイト数 |
| $pp$ | ページポインタのバイト数（＝4バイト） |

NIX のコスト式は，リーフエントリの大きさが1ページより大きいかどうかで異なる．NIX のリーフエントリのサイズ $ll$ は

$$ll=d\times oid+kl+noid \quad \text{if } ll\leq P \quad (18)$$

$$ll=d\times oid+kl+noid+ds \quad \text{if } ll>P \quad (19)$$

で与えられる．$d$ は与えられた要素の値をその集合属性の中に含んでいるデータオブジェクトの総数であり，

$$d=\frac{D_tN}{V} \quad (20)$$

で与えられる．$ds$ は，1エントリがページサイズを越えたときにエントリの先頭に作成されるディレクトリのサイズであり，

$$ds=\left\lceil\frac{d\times oid+kl+noid}{P}\right\rceil\times(oid+pp) \quad (21)$$

☆ 圧縮した BSSF の更新方式の検討は今後の課題とする．

である．ただし，$pp$ はページポインタのサイズである．

集合の定義域の各要素について，その要素を集合属性中に含むオブジェクトが少なくとも一つ存在することを仮定した場合，リーフページの総数 $lp$ は

$$lp=\left\lceil \frac{V}{\left\lfloor \frac{P}{ll} \right\rfloor} \right\rceil \quad \text{if } ll \leq P \tag{22}$$

$$lp=V\times\left\lceil \frac{ll}{P} \right\rceil \quad \text{if } ll > P \tag{23}$$

となる．また，ノンリーフページの総数 $nlp$ は

$$nlp=\left\lceil \frac{lp}{f} \right\rceil+\left\lceil \frac{\left\lceil \frac{lp}{f} \right\rceil}{f} \right\rceil+\cdots+X \tag{24}$$

である．ただし，$f$ はインデックスのファンアウトであり，$X$ は $1\leq X<f$ を満たす整数値である．もし $X$ が1でなければルートページを考慮して $nlp$ に1が加えられる．

$D_t=10$ および $D_t=100$ における $lp$ と $nlp$ の値を**表5**に示す．$N=32,000$ は小規模データベースを想定した場合で，$N=320,000$ は中規模データベースを想定した場合である．$D_t=100$ かつ $N=320,000$ の場合にのみ，リーフノードエントリのサイズが1ページを越え5ページとなり，木の高さもこの場合のみ3となる．残りの場合については木の高さは2である．よって，NIX を用いて一つのキー値を検索する場合のコスト $rc$ は，$D_t=100$，$N=320,000$ の場合には

$rc=3+5=8$（ページ），

その他の3つの場合には

$rc=2+1=3$（ページ）

となる．

**検索コスト**

NIX の検索コストを求めるために，$T\supseteq Q$，$T\subseteq Q$ に対する問い合わせ処理のステップを考える．$T\supseteq Q$ の検索処理では，まず，問い合わせ集合の各要素について，NIX を用いて対応する OID が検索される．この結果，$D_q$ 個の OID の集合が得られる．次に，この $D_q$ 個の OID 集合の積集合がとられる．結果として得られた積集合中の各 OID をもとにデータオブジェクトが検索される．よって，この場合の検索コストは，

$$RC_{\{T\supseteq Q\}}=rc\times D_q+P_sA_{\{T\supseteq Q\}} \tag{25}$$

で与えられる[8]．

$T\subseteq Q$ についても同様に，まず NIX を用いて $D_q$ 個の OID 集合が検索され，次にそれらの OID 集合の和集合がとられる．$T\subseteq Q$ の場合には，NIX の検索により得られた OID に対応するデータオブジェクトが必ずしも検索条件を満たすとは限らないため，シグネチャファイルにおけるフォルスドロップレゾリューションに相当する処理が必要となる．すなわち，和集合中の各 OID について対応するデータオブジェクトが検索され，実際に問い合わせ条件を満たすかどうか調べられる．条件を満たしたデータオブジェクトが問い合わせの結果となる．よって，検索コストは以下のように与えられる[8]．

$$RC_{\{T\subseteq Q\}}=rc\times D_q+P_uN\frac{\sum_{i=1}^{D_t-1}(D_qC_i\times V-D_qC_{D_t-i})}{VC_{D_t}}+P_sN\frac{D_qC_{D_t}}{VC_{D_t}} \tag{26}$$

**記憶コスト**

NIX の記憶コストは

$$SC=lp+nlp \tag{27}$$

である．

**更新コスト**

ノードの分割などを考慮しなければ，集合値の挿入および削除には，NIX に対する $D_t$ 回の挿入/削除が必要となる．よって NIX の更新コストは，

$$UC_I=rc\times D_t \tag{28}$$

$$UC_D=rc\times D_t \tag{29}$$

で与えられる．

**表5** NIX のリーフ／ノンリーフページ数
Table 5 Leaf and non-leaf pages of NIX.

| $N, D_t$ の値 | | $lp$ | $nlp$ |
|---|---|---|---|
| $N=32,000$, | $D_t=10$ | 685 | 5 |
| $N=32,000$, | $D_t=100$ | 6,500 | 32 |
| $N=320,000$, | $D_t=10$ | 6,500 | 32 |
| $N=320,000$, | $D_t=100$ | 65,000 | 304 |

## 4. コストの解析

本章では BSSF と NIX のコストを比較する．$D_t=10$ および $D_t=100$ の場合を対象とするが，$D_t=10$ の場合で，文献8）で述べた内容と重複するものについては適宜省略する．まず，$T\supseteq Q$ の問い合わせに対し，小規模データベース（$N=32,000$）における検索コストの傾向を示し，さらに検索方式の改良とその結果改善された検索コストを示す．次いで，中規模データベース（$N=320,000$）における検索コストを示し，

小規模データベースの場合と比較する．さらに，中規模データベースについては，圧縮したBSSFによる検索コストの改善結果について示す．$T \subseteq Q$ の問い合わせについても同様の議論を行う．最後に記憶コスト，更新コストの評価結果を示す．なお，以下のコスト評価における BSSF のパラメータ $F$ の値は，BSSF の記憶コスト $SC$ が最悪でも NIX と同程度となるように設定する．記憶コストが同等以上という条件のもとで，BSSF の各コストが NIX より優れているのかどうかがコスト評価の焦点となる．

### 4.1 $T \supseteq Q$ の検索コスト

**検索コストの傾向**

例として，小規模データベースを想定した $N=32,000$ の場合における，$D_t=100$，$F=2500$ の BSSF と NIX の検索コストを**図4**に示す．横軸は問い合わせ集合の要素数 $D_q$ であり，最小値は $D_q=1$ である．$m$ としては1から4までの値を用いている．BSSF を用いた集合値検索では，このように比較的小さい $m$ の値を選択した方が検索コストの面で有利であることが文献 8) で示されている．

$D_q=1,2$ などの小さい $D_q$ の値については，(8)式で表されるアクチュアルドロップ数が多くなるため，BSSF，NIX ともに検索コストは大きくなる．BSSF ではこのほかにフォルスドロップも存在し，$m$ と $D_q$ の値が小さい場合にはその影響が現れる．$D_q$ の値がある程度以上になると，検索コストに対するドロップの影響が小さくなり，検索コストの大部分は検索に伴うオーバヘッドのコストとなる．その結果，BSSF と NIX の検索コストは，$D_q$ の増加に伴い単調に増加する．

**検索処理方式の改良**

文献 8) では，より効果的な検索方式として**スマート検索** (*smart retrieval*) を提案した．$D_q=3$ で最小値をとる図4の $m=2$ の BSSF を例にとると，スマート検索の処理は次のようになる．

1. $1 \leq D_q \leq 3$ ならば，通常どおり BSSF を検索する．
2. $D_q \geq 4$ ならば，問い合わせ集合中の任意の三つの要素から問い合わせシグネチャを作り BSSF を検索する．

これは，たとえフォルスドロップが増えても，検索するビットスライスの数を減らした方が，全体の検索コストから見て有利である場合があることに基づいている．ドロップとなったデータオブジェクトが問い合わせ条件を満たすかどうかという最終的な判定が，フォルスドロップレゾリューションの段階で行われることに注意が必要である．これにより，$D_q$ 個の集合要素の一部から問い合わせシグネチャを作った場合でも，フォルスドロップレゾリューションの結果，正しい結果を得ることができる．スマート検索のもとでは，$m=2$ の BSSF の検索コストは $D_q \geq 3$ について一定となる．同様の議論により，NIX についてもスマート検索が適用可能である．

$D_t=100$ の場合のスマート検索に基づく検索コストを**図5**に示す．比較のため $F=1000$ についてもコストを示している．$F=1000$ の BSSF は検索コストでは NIX に劣るが，後述するように記憶コストの面で優れている．$F=2500$ の BSSF については，検索コストがほぼ NIX とほぼ同等となることがわかる．また，$D_q=1$ ($T \ni q$) については NIX が最も優位であることがわかる．図5より，$N=32,000$ については，$T \ni q$ の問い合わせで多少劣るものの，BSSF は NIX

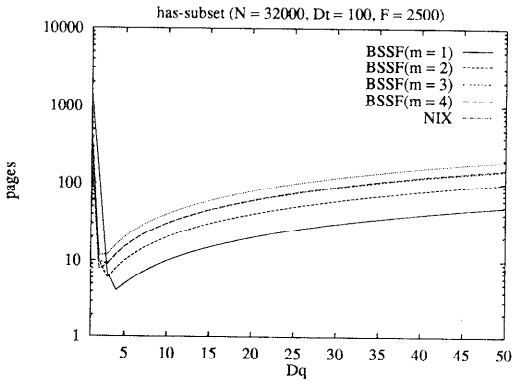



**図4** 検索コスト $RC_{\{T \supseteq Q\}}$ (式(4)，(25)より)
Fig. 4 Retrieval cost $RC_{\{T \supseteq Q\}}$ (Eq. (4), (25)).

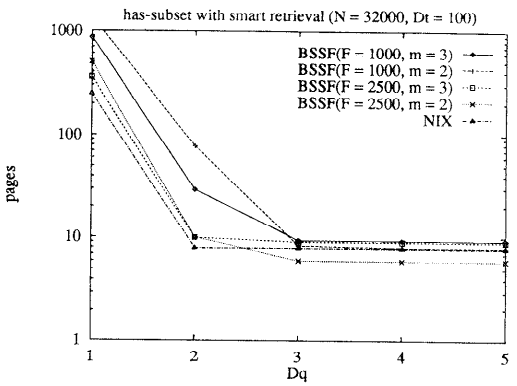



**図5** スマート検索コスト
Fig. 5 Smart retrieval cost.

と同様，$T \supseteq Q$ の集合値検索機構として有望であるといえる．$D_t=10$ の場合についても同様な結果が文献8) で得られている．

**中規模データベースの場合**

$D_t=100$ において，中規模データベースを想定し $N$ を 10 倍の 320,000 とした場合について，スマート検索を用いたときの検索コストを**図6**に示す．BSSF の検索コストが NIX に比べ劣っていることは明らかである．$F=2500$, $m=2$ の BSSF を例にとると，$N=32,000$（図5）では検索コストは NIX と同程度であったのに対し，図6では NIX の数倍のオーダーになっている．これは，ビットスライスの長さが $N$ に比例して増加するため，BSSF の検索コストが $N$ にほぼ比例して増えることによる．一方，NIX の場合には，ある程度の $N$ の値までは $N$ の増加が B-木の構造により吸収されるため，BSSF と比較して $N$ の増加の影響は小さい．$D_t=10$ のグラフは省略するが，この場合についても同様の傾向が見られ，BSSFの検索コストは NIX に大きく劣っている．

**圧縮による検索コストの改善**

中規模データベースに対する検索コストを改善するために，ビットスライスファイルを圧縮した場合の検索コストを考える．圧縮時のビットスライス長は，$SC_{bsf}=\left\lceil \frac{Np(b_t)\bar{n}}{Pb} \right\rceil$ と求められるが(式(13))，参考のため，$N=320,000$, $D_t=100$ において，3種類の $F$ の値についてこの値を求めたものが**表6**である．圧縮を行わない場合の $SC_{bsf}$ が 10 ページであることから，大幅な圧縮が可能であることがわかる．

図6において圧縮を用いた場合の検索コストを**図7**に示す．どのパラメータの設定についても検索コストの向上が見られるが，ビットスライスの“1”の密度が小さくなるような，$F$ が大きく $m$ が小さい場合ほど，圧縮による性能向上が大きいことがわかる．同程度の記憶コストの制約のもとで圧縮した BSSF と圧縮しない BSSF を比べた場合，圧縮した BSSF はより大きい $F$ の値を設定できるため，この点でもより有

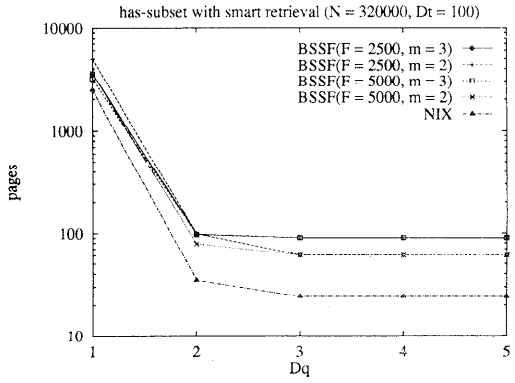



**図 6** 中規模データベースにおけるスマート検索コスト
Fig. 6 Smart retrieval cost for medium-scale database.

**表 6** $\frac{Np(b_t)\bar{n}}{Pb}$ の値（単位：ページ）
($N=320,000$, $D_t=100$)
Table 6 Values of $\frac{Np(b_t)\bar{n}}{Pb}$ (pages).
($N=320,000$, $D_t=100$)

| $F$ | $m=1$ | $m=2$ | $m=3$ | $m=4$ |
|---|---|---|---|---|
| 2500 | 2.72 | 4.59 | 6.11 | 5.94 |
| 5000 | 1.57 | 2.72 | 3.72 | 4.59 |
| 10000 | 0.885 | 1.47 | 2.17 | 2.72 |

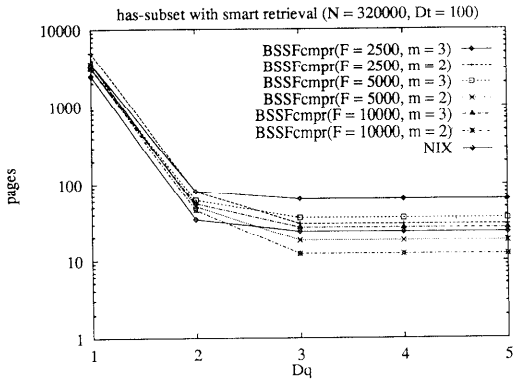



**図 7** 圧縮した BSSF のスマート検索コスト ($D_t=100$)
Fig. 7 Smart retrieval cost of compressed BSSF ($D_t=100$).

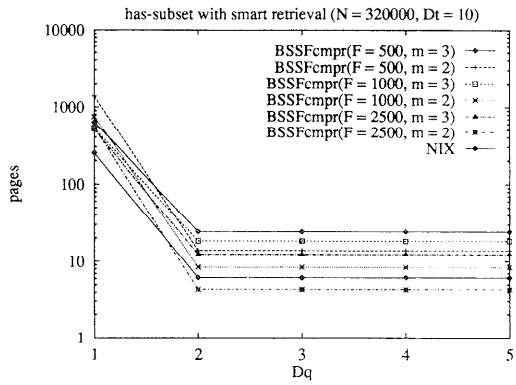



**図 8** 圧縮した BSSF のスマート検索コスト ($D_t=10$)
Fig. 8 Smart retrieval cost of compressed BSSF ($D_t=10$).

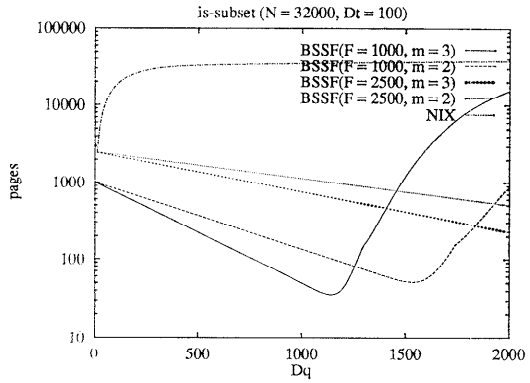



**図 9** 検索コスト $RC_{\{T\subseteq Q\}}$ (式(4), (26)より)
Fig. 9 Retrieval cost $RC_{\{T\subseteq Q\}}$ (Eq. (4), (26)).

利となる．$D_t=10$ の場合の検索コストを**図 8** に示す．$D_q=1$ の場合を除き，パラメータの設定次第で NIX と同程度の検索コストを達成可能であることがわかる．

### 4.2 $T\subseteq Q$ の検索コスト

**検索コストの傾向**

次に，$T\subseteq Q$ の問い合わせに対する検索コスト $RC_{\{T\subseteq Q\}}$ を調べる．まず，検索コストの傾向を示すため，$D_t=100$ における，小規模データベースを想定した $N=32,000$ の場合のコストを**図 9** に示す．この問い合わせでは，$D_q$ が $V$ に非常に近くならない限り，(9)式で与えられるアクチュアルドロップは無視できるほど小さいため，検索コストは主に検索のオーバヘッドとフォルスドロップレゾリューションからなる．

BSSF では，問い合わせシグネチャにおける“0”の値に対応するビットスライスが検索されるが，そのようなビットスライスは $D_q$ が増えるほど少なくなるため，$D_q$ の増加につれスキャンコストは減ることになる．しかし，ある時点からフォルスドロップ確率が急激に大きくなるため，検索コストは再び増えることになる．BSSF のグラフが極小となる $D_q$ の値 $D_q^{opt}$ の式は文献 8) において導かれている．

一方，NIX の検索コストは $D_q$ の増加につれ単調に増加する．NIX の検索では OID 集合の和集合が作られるが，$D_q$ の増加につれ，この和集合の要素数が急速に $N$ に近づくため，ほとんどのオブジェクトが検索されねばならないことが原因である．

**検索処理方式の改良**

BSSF については，$T\supseteq Q$ の場合と同様に $T\subseteq Q$ についてもスマート検索を用いることができる[8]．一方，NIX については BSSF のような検索コストの改良の手段は存在しない．

BSSF に対するスマート検索の概要は以下のようになる．

- 与えられた問い合わせの集合要素数 $D_q$ が $D_q\leq D_q^{opt}$ を満たすなら，$D_q=D_q^{opt}$ の場合に検索するビットスライスの数だけビットスライスを検索し，以降の処理を行う．
- $D_q>D_q^{opt}$ ならば通常どおり検索を行う．

図 9 でもわかるとおり，$D_q^{opt}$ の値は一般に十分大きいため，現実的な $D_q$ の値について，検索コストは $D_q=D_q^{opt}$ における検索コストの値で一定となる．$N=32,000$ の場合の BSSF の検索コストが NIX に比べ優れていることは明らかである．$D_q=10$ に対する同様の結果が文献 8) で示されている．このように，$T\subseteq Q$ の問い合わせについては，NIX は BSSF に比べ検索コストが大きく劣る．

**中規模データベースの場合**

BSSF について，$N=320,000$, $D_t=100$ の場合の $D_q^{opt}$ の値と，$D_q=D_q^{opt}$ における検索コストの値を**表 7** に示す．$D_q=D_q^{opt}$ における BSSF の検索コストは $N$ に比例し，$N=32,000$ の場合のほぼ 10 倍となっている．ちなみに，$N=320,000$, $D_t=100$ の場合，NIX の検索コストは $D_q=100$ の時点ですでに 35,000 ページ程度であり，BSSF に大きく劣っている．同様のことが $D_t=10$ の場合にも成立する．

$T\supseteq Q$ の場合とは異なり，大きな $F$ の値を用いることが検索コストの向上に必ずしも寄与しないことに注意が必要である．$T\subseteq Q$ の場合には問い合わせシグネチャの“0”のビットに対応するビットスライスが検索されるが，$F$ の増加により問い合わせシグネチャ中の“0”のビット数が増えるため，検索すべきビットスライス数が増えることがその理由である．

**表 7** 中規模データベースにおける検索コスト
(単位：ページ)
($T\subseteq Q$, $N=320,000$, $D_t=100$)
Table 7 Retrieval cost for medium-scale database (pages).
($T\subseteq Q$, $N=320,000$, $D_t=100$)

| $F, m$ の値 | $D_q^{opt}$ | 検索コスト |
|---|---|---|
| $F=2500$, $m=2$ | 3960 | 1170 |
| $F=2500$, $m=3$ | 2940 | 818 |
| $F=5000$, $m=2$ | 8130 | 2173 |
| $F=5000$, $m=3$ | 6010 | 1523 |

表 8 圧縮したBSSFの検索コスト（単位：ページ）
($T \subseteq Q$, $N=320,000$, $D_t=100$)
Table 8 Retrieval cost of compressed BSSF (pages).
($T \subseteq Q$, $N=320,000$, $D_t=100$)

| $F, m$ の値 | $D_q^{opt'}$ | 検索コスト |
|---|---|---|
| $F=2500$, $m=2$ | 3870 | 626 |
| $F=2500$, $m=3$ | 2910 | 543 |
| $F=5000$, $m=2$ | 7780 | 739 |
| $F=5000$, $m=3$ | 5830 | 669 |
| $F=10000$, $m=2$ | 11820 | 1926 |
| $F=10000$, $m=3$ | 11660 | 983 |

**圧縮による検索コストの改善**

圧縮を用いた場合には，文献 8) で導いた $D_q^{opt}$ の式は適用できない．しかし，実際にパラメータの値を設定し計算を行うことにより，検索コストを最小とする $D_q$ の値を調べることができる．**表 8** に，$D_t=100$ の場合の，そのようにして得られた検索コストを最小とする $D_q$ の値（$D_q^{opt'}$ と呼ぶ）と，その検索コストの値を示す．

表 7 と表 8 より，同じ $F$ と $m$ の値に関しては圧縮を行うことにより検索コストの大きな向上が得られることが分かる．圧縮による検索コストの向上は $D_t=10$ の場合も同様に確認できる．

### 4.3 記憶コスト

$N=32,000$ の場合の記憶コストと更新コストについては，文献 8) で示した．本稿では，$N=320,000$ の場合の記憶コストを示す．**表 9** は，BSSF と NIX の記憶コストを示したものである．BSSF の記憶コストは NIX の半分，ないし同程度であるが，上に述べたとおり $T \supseteq Q$ の検索コストで大きく劣るため，中規模データベースに対して BSSF が適しているとはいいがたい．

一方，$N=320,000$ における圧縮した BSSF の記憶コストを**表 10** に示す．表 9 の NIX のコストと比べると，圧縮を用いた BSSF の優位性は明らかである．

### 4.4 更新コスト

$N=320,000$, $D_t=100$ の場合について，**表 11** に $D_t=100$ における BSSF と NIX の更新コストを示す．挿入コストについては BSSF が優れている．削除コストについては，$D_t=10$ については NIX が優れ，$D_t=100$ については BSSF が優れている．圧縮を用いた BSSF の更新コストの見積りは今後の課題である．

表 9 BSSFとNIXの記憶コスト（式(14), (27)より）
Table 9 Storage cost of BSSF and NIX.
(Eq. (14), (27))

| $D_t$ | ファイル | ページ数 |
|---|---|---|
| 10 | BSSF ($F=250$) | 3125 |
| | BSSF ($F=500$) | 5625 |
| | NIX | 6532 |
| 100 | BSSF ($F=2500$) | 25625 |
| | BSSF ($F=5000$) | 50625 |
| | NIX | 65304 |

表 10 圧縮した BSSF の記憶コスト（単位：ページ）
($N=320,000$)（式(14), (27)より）
Table 10 Storage cost of compressed BSSF (pages).
($N=320,000$) (Eq. (14), (27))

| $D_t$ | $F$ | $m=2$ | $m=3$ |
|---|---|---|---|
| 10 | 500 | 1987 | 2483 |
| | 1000 | 2194 | 2799 |
| | 2500 | 2461 | 3204 |
| 100 | 2500 | 12099 | 15899 |
| | 5000 | 14242 | 19200 |
| | 10000 | 16311 | 22362 |

表 11 更新コスト（単位：ページ）
($N=320,000$)（式(16), (17), (28), (29)より）
Table 11 Update cost (pages).
($N=320,000$) (Eq. (16), (17), (28), (29))

| $D_t$ | ファイル | $UC_I$ | $UC_D$ |
|---|---|---|---|
| 10 | BSSF ($F=250$, $m=2$) | 20 | 332 |
| | BSSF ($F=250$, $m=3$) | 29 | 341 |
| | BSSF ($F=500$, $m=2$) | 21 | 332 |
| | BSSF ($F=500$, $m=3$) | 30 | 342 |
| | NIX | 300 | 300 |
| 100 | BSSF ($F=2500$, $m=2$) | 193 | 505 |
| | BSSF ($F=2500$, $m=3$) | 284 | 595 |
| | BSSF ($F=5000$, $m=2$) | 197 | 509 |
| | BSSF ($F=5000$, $m=3$) | 292 | 604 |
| | NIX | 800 | 800 |

## 5. おわりに

本論文では，集合値検索を支援する索引機構としてのビットスライストシグネチャファイル (BSSF) と入れ子型インデックスの性能を評価した．問い合わせの種類としては，$T \supseteq Q$, $T \subseteq Q$ の二つを対象とした．本論文では特に，中規模データベースにおける集合値検索機構としての BSSF の有用性を調べることに焦点をあて，検索コスト，記憶コスト，更新コストについて入れ子型インデックスとの比較を行った．BSSF

については，ビットスライスファイルの圧縮を行った場合についてもコストモデルを作成し，その性能を比較した．検索コストに関しては，文献 8) で提案したより効率的な検索方式であるスマート検索に基づくコストについて主に評価を行った．

$T \supseteq Q$ の問い合わせについては，小規模データベース ($N=32,000$) では両者の検索コストはほぼ同程度であった．しかし，中規模データベース ($N=320,000$) では，圧縮を用いない BSSF は入れ子型インデックスに劣ることが明らかとなった．一方，圧縮した BSSF では，BSSF と入れ子型インデックスの検索コストはほぼ同程度となり，ファイル圧縮の効果が見られた．

$T \subseteq Q$ の問い合わせについては，入れ子型インデックスの検索コストは BSSF に比べ大きく劣り，オブジェクトの総数 $N$ が大きくなるとその差はさらに拡大することを示した．この場合にも，ビットスライスファイルの圧縮による BSSF の一層の検索コストの向上が確かめられた．

記憶コストに関しては，ファイルの圧縮の効果は大であり，$T \supseteq Q$ については半分以下の記憶コストでも入れ子型インデックスと同等の検索効率を得ることができることが示された．

以上により，小規模のデータベースについては，BSSF 方式を用い，中規模のデータベースについては圧縮した BSSF 方式を用いることにより，入れ子型インデックスと比較して，$T \supseteq Q$ の問い合わせについては同等の，$T \subseteq Q$ の問い合わせや記憶コストに関してはより優れた結果を得ることができた．これにより，小・中規模のデータベースにおける集合値検索に対するシグネチャファイルの有効性が明らかとなった．

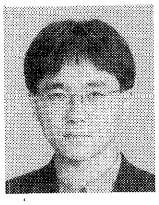
圧縮したビットスライスファイルのサイズの評価式は最適なコード化を想定したものであり，現実の圧縮方式を用いた場合には圧縮効率が低下すると考えられる．今後の課題としては，現実のファイル圧縮方式を想定した場合の BSSF のコスト評価と，その場合の効率的な検索処理方式の開発が挙げられる．また，圧縮した BSSF における効率良い更新方式や，規模を拡大した場合の有効性などについても研究が必要である．

**謝辞** フォルスドロップ確率に関し共に検討いただいた福島慶明氏，および日頃御助言いただいている筑波大学藤原譲教授ならびに鈴木功教授に感謝いたします．また，著者のうち石川は，お世話になっている奈良先端科学技術大学院大学植村俊亮教授ならびに吉川正俊助教授に感謝いたします．

## 参考文献

1) Bertino, E. and Kim, W.: Indexing Techniques for Queries on Nested Objects, *IEEE Trans. on Knowledge and Data Engineering*, Vol. 1, No. 2, pp. 196-214 (1989).
2) Faloutsos, C. and Christodoulakis, S.: Signature Files: An Access Method for Documents and Its Analytical Performance Evaluation, *ACM Trans. Office Inf. Syst.*, Vol. 2, No. 4, pp. 267-288 (1984).
3) Faloutsos, C.: Access Methods for Text, *ACM Comput. Surv.*, Vol. 17, No. 1, pp. 49-74 (1985).
4) Faloutsos, C. and Chen, R.: Fast Text Access Methods for Optical and Large Magnetic Disks: Designs and Performance Comparison, *Proc. Int'l. Conf. on Very Large Data Bases*, Los Angeles, CA, pp. 280-293 (Aug. 1988).
5) Faloutsos, C. and Christodoulakis, S.: Description and Performance Analysis of Signature File Methods for Office Filing, *ACM Trans. Office Inf. Syst.*, Vol. 5, No. 3, pp. 237-257 (1987).
6) 福島慶明，石川佳治，于旭，北川博之，大保信夫：複合オブジェクトに対する索引機構の研究，第 44 回情報処理学会全国大会論文集，2E-1 (1992).
7) Gallager, R. G. and Voorhis, D. C. V.: Optimal Source Codes for Geometrically Distributed Integer Alphabets, *IEEE Trans. on Information Theory*, pp. 228-230 (Mar. 1975).
8) Ishikawa, Y., Kitagawa, H. and Ohbo, N.: Evaluation of Signature Files as Set Access Facilities in OODBs, *Proc. ACM SIGMOD Conf.*, Washington, D. C., pp. 247-256 (May 1993).
9) 石川佳治，北川博之，大保信夫：シグネチャファイルを用いた集合値検索の支援機構，第 46 回情報処理学会全国大会論文集，3F-1 (Mar. 1993).
10) Kim, W., Chou, H.-T. and Banerjee, J.: Operations and Implementation of Complex Objects, *IEEE Trans. Softw. Eng.*, Vol. 14, No. 7, pp. 985-995 (1988).
11) Kitagawa, H., Fukushima, Y., Ishikawa, Y. and Ohbo, N.: Estimation of False Drops in Set-valued Objects Retrieval with Signature Files, in Proc. 4th Intl. Conf. on Foundations Data Organization and Algorithms (FODO), Evanston, IL (Oct. 1993), *Lecture Notes in Computer Science* 730, pp. 146-163, Springer-Verlag, Berlin (1993).

(平成5年 7 月 12 日受付)
(平成6年12月 5 日採録)



**石川 佳治**（正会員）

1965年生．1989年筑波大学第三学群情報学類卒業．1994年同大学大学院博士課程工学研究科単位取得退学．同年より奈良先端科学技術大学院大学情報科学研究科助手．データベースシステムのアーキテクチャ，科学データベースなどに興味を持つ．電子情報通信学会，日本ソフトウェア科学会，ACM, IEEE-CS 各会員．

**北川 博之**（正会員）

1955年生．1978年東京大学理学部物理学科卒業．1980年同大学院理学系研究科修士課程修了．日本電気(株)勤務を経て，1988年筑波大学電子・情報工学系講師．現在同学系助教授．理学博士(東京大学)．データベースシステム構成法，時間データ管理，エンジニアリングデータ管理，ソフトウェア開発支援システムなどに興味を持つ．著書「The Unnormalized Relational Data Model」(共著，Springer-Verlag)．電子情報通信学会，日本ソフトウェア科学会，ACM, IEEE-CS 各会員．

**大保 信夫**（正会員）

昭和20年6月21日生．東京大学理学部卒業．理学博士．筑波大学電子情報工学系勤務．研究テーマ：データベースシステム．ACM, IEEE-CS 学会各会員．