データベースシステム 116－66
（1998．7．10）

ビットコーディングを用いたR-treeに基づく多次元空間内近傍探索の高速化

櫻井 保志[†§] 吉川 正俊[†] 植村 俊亮[†]

†奈良先端科学技術大学院大学 情報科学研究科
奈良県生駒市高山町8916番地の5
{yasusi-s, yosikawa, uemura}@is.aist-nara.ac.jp

§NTTヒューマンインタフェース研究所
神奈川県横須賀市光の丘1-1
sakurai@marsh.hil.ntt.co.jp

あらまし 本論文では，高次元空間内のオブジェクト近傍探索に有用な空間ビットコーディング法と呼ぶインデックス構築手法を提案し，その構造と探索，挿入，削除アルゴリズムについて述べる．画像データベースシステムにおいては，テキスト形式で記述された付加情報に基づく検索のみならず，画像の内容に基づく検索が優れたヒューマンインタフェースを実現する上で必要である．しかし内容検索を実現するには，画像処理を実行して特徴量を抽出した後，類似した特徴量を有する1つもしくは，複数個の画像サンプルをデータベースから探索する必要がある．特にデータベースが大規模化し，さらに特徴量における次元数が高次元化するほど，類似検索のための探索処理が高負荷となる．

これに対し，R-treeおよびその派生手法は対象画像の射影空間位置に基づいて，迅速に類似検索を実行するためのインデックス探索法である．本論文で提案する空間ビットコーディング法はR-treeの技術を利用するとともに，これに範囲矩形(bounding rectangle)の位置，サイズをビットコーディングによって表現する仮想範囲矩形(virtual bounding rectangle)の概念を導入することにより，探索時においてさらなるディスクアクセスの低減化を実現した．

本研究では実験を通じて，探索，挿入処理のディスクアクセス数を計測し，従来手法と比較することにより空間ビットコーディング法を評価する．その性能評価によって，大規模データ集合に対する本手法の優位性を示す．

キーワード 高次元空間，R-tree，仮想範囲矩形，近傍探索，ランク付け問い合わせ

High Dimensional Nearest Neighbor Searching by R-tree Using Bit Coding

Yasushi Sakurai[†§] Masatoshi Yoshikawa[†] Shunsuke Uemura[†]

†Graduate School of Information Science
Nara Institute of Science and Technology
8916-5 Takayama, Ikoma, Nara 630-01 Japan
{yasusi-s, yosikawa, uemura}@is.aist-nara.ac.jp

§NTT Human Interface Laboratories
1-1 Hikarinooka, Yokosuka, Kanagawa
239-0847 Japan
sakurai@marsh.hil.ntt.co.jp

Abstract We present a new indexing method called the Spatial Bit Coding Method which is useful for nearest neighbor searching in high dimensional space, then introduce tree structure and algorithms for searching, insertion and deletion. An image database system has need for not only retrieval using added information described by text but also content based retrieval to achieve excellent human interface. Content based retrieval needs selecting one or some images which has similar feature extracted by image processing from image database. Especially, for high dimensional and large data set, retrieval processing using image feature involves high cost.

The R-tree and its variants are index search method for fast similarity retrieval based on projective location of target image. Our proposed method, the Spatial Bit Coding Method using R-tree technique is introduced idea of the virtual bounding rectangle of which location and size is represented by bit coding. As a result of adoption this method, fewer disk accesses is achieved in search operation.

Finally, measurements of page accesses for search and insert operation and comparison of new and usual method is presented. The performance test prove the superiority of our method for high dimensional data set.

Key words high dimensional space, R-tree, virtual bounding rectangle, nearest neighbor searching, ranking query

1 まえがき

コンピュータネットワークの広がりと，演算能力の優れたコンピュータの開発により，数値，文字データ，さらに画像データについても格納可能なマルチメディアデータベースを実現しようとする要求が高まっている．このようなマルチメディアデータベースに関しては，データベースの大規模化に伴い，これを効率的，且つ迅速に操作する技術が必要となる．特に検索オペレーションに関しては，画像やイラスト等を用いた類似内容検索の実現と，その検索時間の短縮が望まれている．このような要求を実現するため，本研究は，高次元空間におけるオブジェクト近傍探索に適した高速化手法の確立を目的とする．

従来の画像検索アプリケーションは，ユーザインタフェースに優れたものが提案されているが，検索の高速化に対して施しがなされておらず，データベースの大規模化には対応していない[AMP93]．データ集合が大規模化した2次元及び3次元画像アプリケーションにおいては，抽出した特徴に基づいて迅速に画像内容検索を実行するため，インデックス検索法を実装したDBMSが必要である．

一方，画像認識では多くの場合，画像をいくつかの特徴パラメータで表現する．しかしインデックス検索法の中でも，ハッシュテーブルのような値に基づいて完全照合を行う方法は，多次元空間上で表現されるオブジェクトに対応しておらず，画像データベースには不向きである．これはB+-treeについても同様である[KS91]．

多次元空間における範囲探索に関しては，R-tree[Gut84]，もしくはその派生手法がこれを効率的に実施する構造として，非常に有用である．R-treeのバリエーションの1つであるR*-tree[BKSS90]は，R-treeの挿入手法を改良し，探索性能の向上を図ったものであり，多次元空間探索において最も標準的に用いられている手法の1つである．

R-treeとその派生手法はいずれも，範囲矩形を表現するために厳密な座標位置を用いている．これは探索処理において多くのコストを要する．これに対し，本論文では空間ビットコーディング法を提案する．空間ビットコーディング法は，幾何学的オブジェクトの位置，サイズを各次元あたり数ビットで表現する．このことにより，探索においては従来と比較して少量のディスクアクセスで近傍オブジェクトを検出することが可能となる．後述する評価実験では，実際に本手法が従来手法を上回る性能を有することを明らかにしている．

本論文の構成は以下の通りである．2節では，R-treeとR*-treeのインデックス構造，さらにその構造を用いた近傍探索手法について記述する．3節は，空間ビットコーディング法に関する記述である．4節において，空間ビットコーディング法を用いた実験結果を提示する．5節は，結論とまとめである．

2 関連研究

空間アクセス法の中で，最も有望なアプローチの1つとしてR-tree[Gut84]が挙げられる．R-treeは空間内の幾何学的オブジェクトを最小範囲矩形(以下，MBRと呼ぶ)に梱包し，そのオブジェクトを包含したMBRをリーフとして木状に階層化した構造を持つ．リーフノードは以下のエントリを含んでいる．

$$\begin{aligned} LeafNode &= (\varepsilon, Record_i)\ (i = 1, \ldots, \varepsilon) \\ Record_i &= (C, object-identifier) \end{aligned}$$

ここで，εはリーフノードに格納されているエントリ数であり，下限値mと上限値Mに制限される$(m \leq \varepsilon \leq M)$．$C$は幾何学的オブジェクトを近似したMBR，もしくは点で表現されるオブジェクトの空間ベクトルを表すものであり，具体的にはn次元空間上の座標値を保持する．また，$object-identifier$によってオブジェクトのコンテンツにアクセスすることができる．さらに，ノンリーフノードは以下のエントリを含むものである．

$$\begin{aligned} NonleafNode &= (\varepsilon, Record_i)\ (i = 1, \ldots, \varepsilon) \\ Record_i &= (C, child-pointer) \end{aligned}$$

Cは，子ノードに格納されている矩形の全てを包含するMBRを表している．また，child-pointerによって子ノードにアクセスすることができる．R*-tree[BKSS90]も上記と同様の構造であるが，再挿入処理(forced reinsert)等，新たな挿入処理アルゴリズムを導入することにより探索性能の向上を実現している．

R-tree構造を用いた近傍探索手法に関しては，Roussopoulosらによって提案されている[RKV95]．この手法は探索順序付け(ordering)および枝刈り(pruning)を用いることにより，オブジェクト探索を失策すること無く，低コストで探索するものである．最近傍探索において，MBRの枝刈りは3つの戦略に基づいて実施される．下式では，Qが問い合わせ点，R, R'がMBR，O, O'がオブジェクトを表しており，式が成立した時点でRとOが枝刈り対象として除去される．

$$\begin{aligned} Strategy1 &: MINDIST(Q,R) > MINMAXDIST(Q,R') \\ Strategy2 &: \|Q,O\| > MINMAXDIST(Q,R') \\ Strategy3 &: MINDIST(Q,R) > \|Q,O'\| \\ & \quad (R \neq R', O \notin R', R \not\ni O') \end{aligned}$$

ここで，$MINDIST(a,A)$は点aと矩形Aの最小距離，$\|a,b\|$は点ab間のユークリッド距離を表している．また$MINMAXDIST(a,A)$とは，$MINDIST(a,A)$を求める際，可能性として最大となる値である．具体的には，n次元空間中のAを構成するn個の超平面各々において，aとの距離が最大となる頂点を$b_i(i = 1, \ldots, n)$

としたとき，$\min_i \|Q, b_i\|$ を指す．このような 3 つの戦略を組み合わせ，ディスクアクセスの低減を図っている．

[RKV95] では，主として最近傍探索をターゲットとしている．本論文では，k個の近傍を探索するランク付け問い合わせ (ranking query) の実現を目的としているため，*Strategy*1 と *Strategy*2 を以下のように拡張して解釈する．

$$\begin{aligned} Strategy1' &: MINDIST(Q,R) > D \\ Strategy2' &: \|Q,O\| > D \\ & (R \neq R', O \notin R') \end{aligned}$$

ここで，

$$D = \begin{cases} NN.dist[k] \\ \quad if\ NN.dist[k] \leq MINMAXDIST(Q,R') \\ \quad\quad \wedge NN.number = k \\ MINMAXDIST(Q,R') \quad otherwise. \end{cases}$$

$NN.number$は探索処理により収集したオブジェクトの数である．また，$NN.dist[i](0 \leq i \leq NN.number)$ は収集オブジェクトに対応した近傍距離である．収集したオブジェクト数が要求される k個に達しており，さらにその数が $MINMAXDIST(Q,R')$ よりも小さければ，k番目のオブジェクトと問い合わせ点との距離を用いる．それ以外の場合のみ [RKV95] らの提案する戦略を適用することになる．後述の性能試験においても，*Strategy*1 を包含する *Strategy*1′と，*Strategy*2 を包含する *Strategy*2′を用いる．

3 空間ビットコーディング法

本節では空間ビットコーディング法の構造および探索，挿入，削除手法について述べる．これまで，R-tree バリエーションに属する手法の議論は，リーフノード，ノンリーフノード各々における MBR のクラスタリングに関するものが中心であった．本論文で提案する空間ビットコーディング法は，これらの議論とは一線を画すものである．本手法では，範囲矩形 (bounding rectangle) の位置，サイズを各次元あたり数ビットで表現し，そのビットで表現された仮想範囲矩形 (virtual bounding rectangle) を用いて探索を行っている．このことにより，高次元空間におけるさらなる探索性能の向上が可能となる．

3.1 空間ビットと仮想範囲矩形

n 次元空間における範囲矩形は，n の座標軸各々に対応して始点 (initial vertex) と終点 (terminal vertex) の座標値を有している．これは，空間における絶対的位置と言える．絶対的位置を用いれば，厳密な位置表現が可能であるが，反面多くの表現コストを必要とする．

これに対して，空間ビットは上位ノードの座標値を利用し，相対的位置を表現するものである．

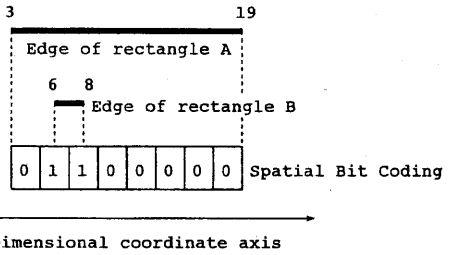

図 1: 空間ビットコーディング (8 bit coding)

定義 1 n 次元空間内において，頂点 a, a'が対角をなす範囲矩形 A と頂点 b, b'が対角をなす範囲矩形 Bが存在する．符号長を $n \cdot l$ (次元あたり lビット，合計 $n \cdot l$) とすると，下記関数 *genSpatialBit*(A,B) により生成されるビットを"A による Bの空間ビット"と呼ぶ．

$$genSpatialBit(A,B) = [s_{ij}]_{n \times l},\ B \subseteq A$$
$$(i = 1,2,\ldots,n;\ j = 1,2,\ldots,l)$$

ここで，

$$s_{ij} = \begin{cases} 1 & if\ u_{j-1} \leq \psi_i' \wedge u_j > \psi_i \\ & \quad \vee u_j = \psi_i = \psi_i' \wedge j = l \\ 0 & otherwise. \end{cases}$$

$$u_k = \phi_i + k \cdot (\phi_i' - \phi_i)/l$$
$$a = [\phi_1, \phi_2, \ldots, \phi_n],\ a' = [\phi_1', \phi_2', \ldots, \phi_n']$$
$$b = [\psi_1, \psi_2, \ldots, \psi_n],\ b' = [\psi_1', \psi_2', \ldots, \psi_n']$$
$$\phi_i \leq \phi_i',\ \psi_i \leq \psi_i'$$

例 1 図 *1*のように n 次元空間内の i 次元座標軸において，範囲矩形 A が座標値として *(3, 19)* を，そして範囲矩形 Bが同様に座標値として *(6, 8)* を占めているものとする．A の領域を *8* 分割した場合，Bは A の始点から数えて前から *2* 番目と *3* 番目の領域を占めていることになる．符号長 $l = 8$ とすると，定義 *1*より A と Bの座標値を用いて，Bの領域を *'01100000'* のように符号化することが可能である．

少ない情報量にもかかわらず，空間ビットを用いて範囲矩形を復元することも可能である．ただし，復元された範囲矩形は誤差を含んでいる．絶対的な位置表現をしている元の範囲矩形に対し，空間ビットから復元する相対的な位置表現の矩形を仮想範囲矩形 (virtual bounding rectangle，もしくは VBR) と呼ぶ．

定義 2 定義 *1*において生成された空間ビットを Sとするとき，下記関数 *genVirtualBR(A, S)* により生成される矩形 Vを"A による Bの仮想範囲矩形"と呼ぶ．

$$V = genVirtualBR(A,S) = (v, v')$$

ここで，

$$v=[\tau_1,\tau_2,\ldots,\tau_n],\ v'=[\tau'_1,\tau'_2,\ldots,\tau'_n],\ \tau_i\le\tau'_i$$
$$\tau_i=\sum_{j=1}^{l-1}f(i,j),\ \tau'_i=\sum_{j=1}^{l-1}g(i,j)$$

$$f(i,j)=\begin{cases}\phi_i & if\ s_{ij}=1\wedge j=1\\ \phi_i+j\cdot(\phi'_i-\phi_i)/l & \\ \qquad if\ s_{ij}=0\wedge s_{i(j+1)}=1 & \\ 0 \quad otherwise. & \end{cases}$$

$$g(i,j)=\begin{cases}\phi_i+j\cdot(\phi'_i-\phi_i)/l & \\ \qquad if\ s_{ij}=1\wedge s_{i(j+1)}=0 & \\ \phi_i+(j+1)\cdot(\phi'_i-\phi_i)/l & \\ \qquad if\ s_{i(j+1)}=1\wedge j=l-1 & \\ 0 \quad otherwise. & \end{cases}$$
$$(i=1,2,\ldots,n;\ j=1,2,\ldots,l-1)$$

例 2 例 1が 1次元であったのに対し，2次元空間に空間ビットを適用した例を考える．図 2のように $B\subseteq A$ となる範囲矩形 A と範囲矩形 Bが 2次元空間上に配置されているものとする．1次元，2次元の座標値を定義 1において示した関数 *genSpatialBit(A, B)* に適用すると，Bの空間ビット *'01100110'* が得られる．これを用い，図 2中の点線で示す矩形を定義 2により復元することができる．この点線で示される矩形は Bの *VBR* を示している．

定理 1 範囲矩形 Bと Bの *VBR* Vにおいて，$B\subseteq V$ が成り立つ．

証明: 定義1において，$S_{ij}=1, S_{i(j+1)}=0, S_{i(j+2)}=1$ となる場合は有り得ない．従って，定義2において $S_{i1}=1$ のとき，$\tau_i=\phi_i$であるため$\tau_i\le\psi_i$．さらに，定義2において $S_{ik}=0, S_{i(k+1)}=1$ のとき，

$$\tau_i=\phi_i+k\cdot(\phi'_i-\phi_i)/l$$

である．このとき定義1から

$$\phi_i+k\cdot(\phi'_i-\phi_i)/l\le\psi_i<\phi_i+(k+1)\cdot(\phi'_i-\phi_i)/l$$

であるため，$\tau_i\le\psi_i$．

また，定義2において $S_{il}=1$ のとき，$\tau'_i=\phi'_i$であるため$\tau'_i\ge\psi'_i$．さらに，定義2において $S_{ik}=1, S_{i(k+1)}=0$ のとき，

$$\tau'_i=\phi_i+k\cdot(\phi'_i-\phi_i)/l$$

である．このとき定義1から

$$\phi_i+(k-1)\cdot(\phi'_i-\phi_i)/l\le\psi'_i<\phi_i+k\cdot(\phi'_i-\phi_i)/l$$

であるため，$\tau'_i>\psi'_i$．

従って，$\tau_i\le\psi_i, \tau'_i\ge\psi'_i$ であるため，$B\subseteq V$が成立する．

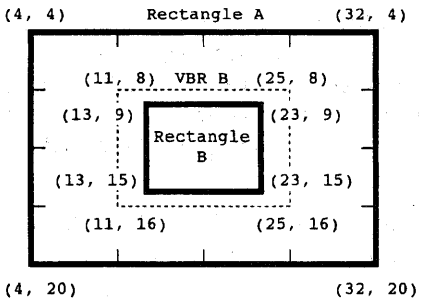

図 2: VBR による空間位置表現 (4*2 bit coding)

補題 1 *[RKV95]* における *Strategy3* は，*VBR* Vに対して適用することができる．

証明: 定理 1より，任意の問い合わせ点 Q に対して，MBR Bと Bの VBR Vの間に下式が成り立つ．

$$MINDIST(Q,V)\le MINDIST(Q,B)$$

従って，*Strategy3'*を探索処理に適用することができる．

$$Strategy3':\ MINDIST(Q,V)>\|Q,O'\|$$

3.2 空間ビットの木構造への適用

空間ビットコーディング法を木構造へ適用した場合，以下の点で従来と異なる．

(1) 本手法を適用する木構造とは別に，空間ビットで構成される木構造を生成する．2つの木構造を区別するために，絶対的位置表現を用いる元の木構造を実部分 (real part)，相対的位置表現を用いる空間ビットにより構成される木構造を仮想部分 (virtual part) と呼ぶ．

(2) 木構造の実部分における各々のノードに対応して，仮想部分のノードを生成する．ただし，実部分のリーフノードについては，仮想部分において生成されない．

(3) 実部分の木構造の高さが s であるとき，仮想部分の $s-1$ レベルのノードは，子ノードのポインタとして実部分の s レベルのノード，つまりリーフノードを指す．

(4) 根ノードの MBR の位置，大きさを座標値として保持しておく必要がある．

(5) 仮想部分において子ノードの VBR を表現する空間ビットは，現ノードの VBR と実部分に含まれる子ノードの MBR を基に生成される．ただし，現ノードが根である場合のみ，根の MBR と子ノードの MBR を基に子ノードの VBR が生成される．

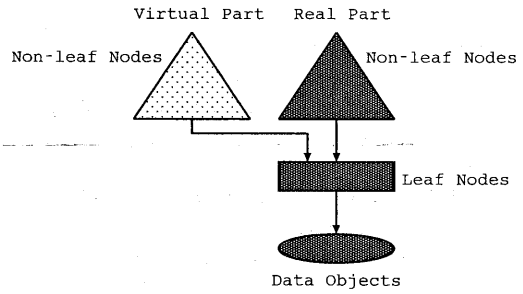

図 3: 木構造における実部分と仮想部分

(6) 探索処理においては，仮想部分の各ノード毎に，現ノードの VBR と子ノードの空間ビットを基に子ノードの VBR を生成する．その VBR に対して枝刈り戦略を適用する．

図 3は，実部分と仮想部分の関係を表したものである．

空間ビットコーディング法では，実部分とは別に仮想部分を生成する．仮想部分のノードは以下のような構造を有する．

$$Node = (\varepsilon, S, child-pointer_i)\ (i = 1, \ldots, \varepsilon)$$

$$child-pointer_i = (page-number, byte-offset)$$

ここで，εはリーフノードに格納されているエントリ数である．Sは，ε個の子ノードの MBR を空間ビットで表現したものである．また，$child-pointer_i$によって子ノードにアクセスすることができる．$child-pointer_i$の中にはページ番号の他にバイトオフセット [Dat85] が格納されている．R-tree においては，主として 1 ノードが 1 ページを占有するように設計されているが，本手法の仮想部分では 1 ページに複数のノードが格納されるため，ページ内でレコードが格納されているバイト変位を $byte-offset$ として示している．

3.3 探索処理

探索処理においては，実部分のリーフノードと仮想部分を用いて処理を行う．補題 1で示したように，VBR を [RKV95] で示されているアルゴリズムに適用することが可能である．ただ，VBR による探索を実施するためには，戦略を以下のように修正する必要がある．

(1) *Strategy1′*を適用することはできない．何故なら，VBR には [RKV95] における **Lemma 2** と **Theorem 2** が成り立たないためである．

(2) *Strategy3* の代わりに，補題 1において示した *Strategy3′*を適用する．また，*Strategy2′*に関しては，MBR を VBR に読み替えて適用する．

図 4は，VBR を用いた近傍探索アルゴリズムである．木構造を用い，点 Q を中心とした近傍オブジェクトを k 個収集する．再帰プロシージャ *collectEntry*では，定義 2において示した*genVirtualBR*関数を用いている．経由したノード毎に，パラメータとして引き継いだ範囲矩形と子ノードの空間ビットに基づいて，各子ノードの VBR Vを生成する．Vによって，枝刈り対象の判断を行う．枝刈り処理に関しては，*downwardPruning*が *Strategy2′*を，*upwardPruning*が *Strategy3′*を実行する関数である．枝刈り対象にならなかった子ノードについては，子ノードの VBR をパラメータとして再度 *collectEntry*を実行することにより，孫ノードを探索する．

また，このアルゴリズムにおいて，レベル 1 である場合のみ，根ノードの MBR (*RootRectangle*) をパラメータとして用いる．

```
procedure NNsearch(point Q, int k)
/* "Q" means query point for "k"th nearest neighbor
     search using Virtual part of tree structure */
collectEntry(node VirtualRoot, rect RootRectangle,
                                          level 1);
ReportEntry;

procedure collectEntry(node N, rect R, level L)
/* N is leaf node */
if L = height_of_tree then
  /* Access real-part of tree structure */
  for i:=1 to N.count
  do
    reportMINDIST(Q, N.branch[i].RealRect);
  end;
/* N is non-leaf node */
else
  /* Access virtual-part of tree structure */
  for i:=1 to N.count
  do
    /* Generate virtual bounding rectangle */
    V: = genVirtualBR(R, N.branch[i].SpatialBit);
    /* Set branch list and sort */
    setBranchList(BranchList, N.branch[i].Node, V);
  end;
  downwardPruning(BranchList);
  for i:=1 to BranchList.count
  do
    /* Recursively perform procedure */
    collectEntry(BranchList.list[i].Node,
                 BranchList.list[i].VirtualRect, L+1);
  end;
  upwardPruning(BranchList);
endif;
```

図 4: 近傍探索アルゴリズム

3.4 挿入処理

空間ビットコーディング法の挿入処理は，木構造の実部分のみならず仮想部分についても更新する必要がある．実部分の更新は，R-tree バリエーションで実施している方法と同じである．また，仮想部分におけるノードのオーバーフロー処理は実部分の処理と同一であり，実部分の更新操作を模倣する形となる．実部分におけるオブジェクトの挿入，実部分と仮想部分におけるノードのオーバーフロー処理が全て終了した時点で，仮想部分における空間ビット更新処理を開始する．図5は，挿入処理が発生した際の空間ビット更新アルゴリズムである．*adjustSpatialBit* 関数では，定義1において示した *genSpatialBit* 関数を用い，上位ノードからパラメータとして引き継いだ矩形と現ノードが保有する子ノードの MBR から空間ビットを生成し，更新する．

実部分で MBR を更新したノードについては，対応する仮想部分のノードと，その配下に位置する全てのノードにアクセスし，ノードが保有する空間ビットを検査する．そして，必要あらば更新する．実部分における MBR の修正処理は，上向きに伝播し，修正の必要のないノードに達したときに終了する．これに対して，仮想部分における空間ビットの修正は下向きに伝播する．また，空間ビットに修正を要しないノードに達しても，引き続き下位ノードへ処理を伝播させる．何故なら，空間ビットは相対的位置を用いているために VBR には幾分かの誤差が含まれており，現ノードにおいて修正の必要が無い場合でも下位ノードでは修正を必要とする可能性があるためである．仮想部分における空間ビットを修正するためには，実部分の MBR を参照する必要があることから，空間ビットの修正処理は，仮想部分のみならず，実部分に対してもアクセスすることになる．空間ビットコーディング法の挿入処理は，これを用いない場合と比べ多くのコストを必要とする．

3.5 削除処理

削除処理においても，挿入処理と同様に木構造の実部分のみならず仮想部分についても更新する必要がある．実部分におけるオブジェクトの削除，実部分と仮想部分におけるノードのアンダーフロー処理が全て終了した時点で，仮想部分における空間ビット更新処理を開始する．空間ビットの修正は，挿入処理と同じ図5のアルゴリズムを用いる．

挿入，削除処理では上述のように木構造における実部分と仮想部分の更新操作が発生するため，多くのコストを必要とする．しかし，探索においては，容量の小さい仮想部分を用いて枝刈りを行うことによって，少ないディスクアクセスで処理を行うことを見込んでいる．次節では，具体的な数値を提示して議論する．

```
procedure insert(insert_path I)
/* Renew virtual part of tree structure
                    using insertion path "I" */
for i:=1 to I.count
do
   VNode := VirtualRoot; RNode := RealRoot;
   R := RootRectangle;
   for j:=1 to I.path[i].level - 1
   do
      route := I.path[i].route[j];
      V := genVirtualBR(R,
            VNode.branch[route].SpatialBit);
      VNode := VNode.branch[route].Node; R := V;
      RNode := RNode.branch[route].Node;
   end;
   adjustSpatialBit(VNode, R, I.path[i].level, RNode);
end;

procedure adjustSpatialBit(node VNode, rect R,
                           level L, node RNode)
for i:=1 to VNode.count
do
   S := genSpatialBit(R, RNode.branch[i].RealRect);
   VNode.branch[i].SpatialBit := S;
   if L < height_of_tree - 1 then
      V := genVirtualBR(R, S);
      adjustSpatialBit(VNode.branch[i].Node, V,
                  L+1, RNode.branch[i].Node);
   endif;
end;
```

図 5: 挿入アルゴリズム

4 性能試験

空間ビットコーディング法の効果を検証するために，アルゴリズムを実装し，実験を行った．比較対象として R*-tree を用いる．また，空間ビットコーディング法を適用した木構造において，その実部分についても R*-tree を用いる．また，ノードを経由する際等に発生する CPU 処理時間は，ディスクアクセスに要する時間を大きく下回るものと考え，評価対象から除外した．従って，探索，挿入操作における性能評価はディスクアクセス数と同等として提示する．実験環境については以下の通りである．

- アルゴリズムは UNIX マシン上で C によって実装した．
- データオブジェクトは，ランダム関数による点データを用いる．10K から 100K までのデータ集合サイズに対して実験を実施した．
- 空間次元数は 16 である．

- ディスクページは1K(1024 byte)を基にしている.
- 空間ビットコーディング法を適用した木構造に関しては,次元あたり4ビット,8ビット,12ビットの空間ビットを用いた3種類のインデックスを構築した.
- 木構造における子ノードへのポインタは4バイトとする.ただし,仮想部分のポインタに関してはバイトオフセットが含まれているため,5バイトとする.
- 探索処理においては,10位から50位までのランク付け問い合わせを実施する.
- 探索性能の数値は,問い合わせ施行100回の平均値である.問い合わせ点は,データ集合とは別にランダム関数で生成したデータを用いている.
- 挿入コストは,10Kから100Kまでの各データ集合に対して,1000個のオブジェクト挿入に要するコストの平均値を示している.

4.1 探索コスト

本節では,本手法の探索性能について示す.図6は,10Kから100Kのデータ集合を用いたときの探索コストを従来手法と比較したものである.探索数は30,空間ビットのビット数は最も性能の良い8ビットを用いている.10Kから100Kの全サイズに対して,本手法の優位性が確認できる.例えば100Kのサイズにおいては,ディスクアクセスがR*-treeと比べ,約13.5%下回っている.

本手法の優位性は以下の理由による.探索で用いる仮想部分においては,矩形位置,サイズの表現コストが格段に小さく,ノンリーフノードの容量も小さい.1つのディスクページに複数のノンリーフノードを格納しているため,s個のノンリーフノードを経由しても,s回未満のディスクページのアクセスで事足りる場合がある.その場合,ディスクアクセスの節約になり,性能向上が可能となる.次節では,ディスクアクセス節約の様子をより詳細に説明する.

4.2 探索数のコストに対する影響

図7は,探索数が10から50まで変動させたときのコストを表しており,100Kのデータ集合サイズを用いたものである.またR*-treeの他に,空間ビットのビット数を4,8,12に設定した際のコストを示している.図8は同様の条件下におけるリーフノードのアクセス数を示している.

探索数が多くなると,全てのノンリーフノードの中で,アクセスするノードの割合が高くなる.その結果として図7が示すように,アクセスするノードの割合が高い程,節約の効果が顕著に現れている.

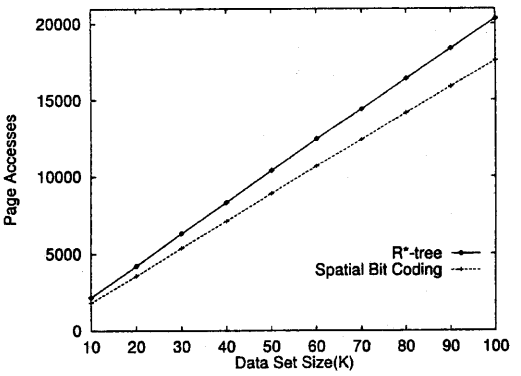

図 6: データ集合サイズに対する探索コスト変動

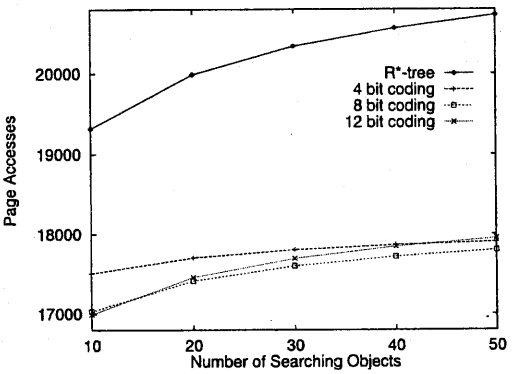

図 7: 探索数に対するコスト変動

また探索数が多い場合,空間ビットの小さい木構造が有利であることが分かる.この事象は以下の理由による.空間ビットのビット数を少なくすることにより,矩形位置,サイズの表現コストを抑制することができる.これは節約効果,探索性能向上の要因となる.一方,空間ビットによって生成されるVBRは,幾分かの誤差を含んでおり,これはビット数を小さくする程,誤差は大きくなる.この誤差は,探索性能低下の要因となる.図8においては,探索数を大きくする程,アクセスするリーフノード数の差が少なくなっていることが分かる.つまり,アクセスするノードの割合が高くなる程,VBRの誤差がノードアクセス数に与える影響が少なくなっていることを示している.この結果,表現コストの抑制効果がより強まり,空間ビットの小さい木構造の利得が大きくなる.

また探索数30の場合,差が微量であるものの,ビット数8の木構造が最良の性能を示している.VBRの誤差と表現コストのバランスが最も良くとれているため,好ましい性能を示している.

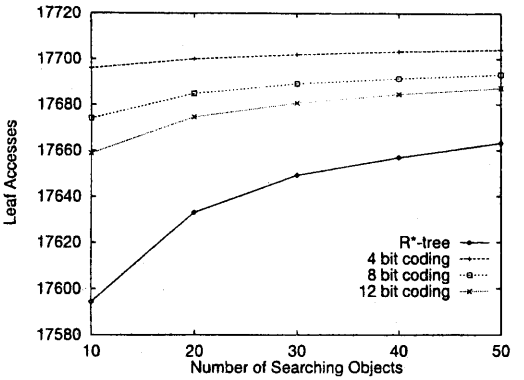

図 8: 探索数に対するリーフノードアクセス数の変動

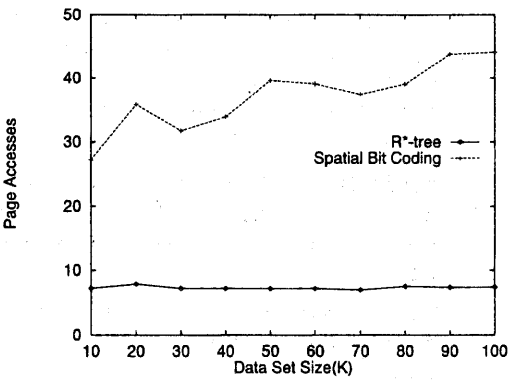

図 9: 挿入コスト

4.3 挿入コスト

木構築のための挿入コストについて議論する．図 9は，挿入コストに関して，従来手法と 8 ビットの空間ビットを用いた本手法とを比較したものである．少ない挿入コストを示す R*-tree に対して，本手法は多くのコストを要する．ノード分割が発生しない場合，R-tree バリエーションでは木の高さを h とすると，挿入コストは最大でも $h+1$ である．これはページ 0 のアクセスを除くと，1 つの挿入パス上のノードしかアクセスしないためである．これに対して本手法では，仮想部分の更新のためのアクセス，さらに空間ビットの正誤を確認するために実部分のノンリーフノードに対するアクセスを必要とする．これは，$h+1$ には納まらない．さらにノード分割が発生し，その分割が上位に伝播した際には，木構造全体へアクセスが起こるため，非常に多くのコストを必要とする．

ただ木構造の構築にあたり，データ蓄積コストに関しては殆んどコスト上昇につながらない．同様の条件で 100K のデータ集合においては，約 3%の上昇に過ぎない．これは実部分のリーフノードに対応するノードを仮想部分が保有しないこと，さらに仮想部分のノードは 1 ページに複数格納されているためである．

5 まとめ

本論文では，高次元空間探索において有効な空間ビットコーディング法を提案した．まず，上位ノードの座標位置を利用して矩形の相対的位置を表現する空間ビットと，その空間ビットを用いて形成される VBR(仮想範囲矩形)を定義した．さらに VBR に基づく近傍探索アルゴリズムを提案した．本手法は，空間ビットを用いることにより位置表現コストを低減させ，探索性能の向上を図ったものである．16 次元空間内に一様分布する 100K のランダムデータ集合を用いた実験では，R*-tree と比較して，約 13.5%の優位性があることを示した．

またデータ蓄積コストに関しては，従来手法と比べ約 3%程度の上昇に過ぎないため，探索性能の優位性を示す本手法が実用面でも適していると考えられる．特に，マルチメディアデータベースを実現する際には有用な技術である．

参考文献

[AMP93] A.D.Bimbo, M.Campanai, and P.Nesi: "A Three-Dimensional Iconic Environment for Image Database Querying", *IEEE Trans. Software Engineering*, Vol. 19, No. 10, pp. 997–1011, 1993.

[BKSS90] N. Beckmann, H. P. Kriegel, R. Schneider, and B. Seeger: "The R*-tree: An Efficient and Robust Access Method for Points and Rectangles", in *Proc. ACM SIGMOD Conf.*, p. 322, Atlantic City, NJ, May 1990.

[Dat85] C. J. Date: *An Introduction to Database Systems (4th Ed.)*, Addison-Wesley, 1985.

[Gut84] A. Guttman: "R-Trees: A Dynamic Index Structure for Spatial Searching", in *Proc. ACM SIGMOD Conf.*, p. 47, Boston, MA, June 1984, Reprinted in M. Stonebraker, Readings in Database Sys., Morgan Kaufmann, San Mateo, CA, 1988.

[KS91] Henry F. Korth and Abraham Silberschatz: *Database System Concepts*, McGraw-Hill, 2nd edition, 1991.

[RKV95] N. Roussopoulos, S. Kelley, and F. Vincent: "Nearest Neighbor Queries", in *Proc. ACM SIGMOD International Conference on Management of Data*, May 1995.