データベースシステム　127－9
情　報　学　基　礎　　67－9
（２００２．５．２１）

規則的球配置と方向に基づく近似を特徴とする多次元インデクス方式

山根　康男
富士通研究所
ドキュメント研究部

三末　和男
富士通研究所
ドキュメント研究部

概要　高次元で問題となるクラスタ分割を確実に行える規則的な球配置と球内空間の方向による極座標的な近似を特徴とする多次元インデクス方式を報告する。球は空間を被覆すると同時になるべく重複がないような位置に正単体（n 次元での正三角形）を基準として配置する。球内空間の近似を直交座標による方法で行うと無駄が生じることを示し、方向の量子化において再帰的な次元分割と適合的なビット割り当てを特徴とする方法を示す。実現容易性を考慮し、ＲＤＢの上にレコードベースで作成することを前提にしている。主記憶上でのシミュレーション実験の結果では、レコードアクセス回数に関して SR-tree の回数を大きく削減した。

A Multidimensional index method characterized by regular positioning of spheres and the approximation based on directions

Yasuo Yamane
Fujitsu Laboratories Ltd.
Document Laboratory

Kazuo Misue
Fujitsu Laboratories Ltd.
Document Laboratory

Abstract: We report a multidimensional index method characterized by regular positioning which is capable of dividing any cluster certainly, which is a problem in high-dimensional spaces, and the approximation method for the inside space of a sphere using directions which is similar to spherical coordinates. Spheres are positioned so that they may cover space and cause as small overlap as possible on the basis of a regular simplex (triangle in n-th dimension). We show the approximation of the inside space of a sphere by orthogonal coordinates cause waste, and present a method by which dimensions are divided recursively and bits are assigned adaptively in quantizing directions. For easiness of implementation, we developed our method on the assumption that it is implemented on RDB in record basis. In the results of simulation, our method largely reduces the total record accesses of SR-tree.

１．はじめに

マルチメディアデータがインタネット、携帯電話、入力装置（スキャナ、ディジタルカメラ）の普及に伴いその件数、量ともに急激に増加しつつある。数十万件から数百万件の画像を管理するケースも珍しくなくなってきている。 件数が多くなると必然的に検索する技術が要求される。メタデータによる検索も可能であるが、人手を介する場合が多いため、内容に基づく類似検索に期待が高まっている。

一方、[Chaudhuri00]で指摘されているように、データベースシステムが長い間の機能追加で複雑化し、機能追加にコストと時間がかかるようになってきている。 これらの状況に鑑み、類似検索において次の２点が重要と考えている。

(1) 高速性

特に、大量データに対する高速性と高次元での高速性が重要な課題である。上記のように、マルチメディアデータの急激な増加に伴い、大量データに対応できる性能が要求されている。しかし、高次元では、次元の呪い(curse of dimensionality)と呼ばれる現象が起き、距離の差が少なくなり、クラスタ分割がしにくくなるという問題が指摘されており[Katayama01]、この問題に取り組む必要がある。

(2) 実現の容易性

[Chaudhuri00]では、データベースの RISC 化が提案されている。また、拡張データベースの研究も多くなされている[Carey96]。ただし、現状では、新しい技術をデータベースの中核部分に組み込むには、時間とコストがか

かるのが実情である。クラスタリングを直接制御しようとすると、ページベースでの方式となり、この組み込みの方法を取らざるをえない。一方、アプリケーションのように、データベースシステムの上に多次元インデクスを構築すれば、実現は容易となる。この場合、レコードベースとなり、直接ページの制御を行なうことはできない。したがって、高速性実現のためには、データベースシステムによるクラスタリングを考慮しつつ、レコードへのアクセス回数を減らすことが重要な課題となる。

多次元インデクスに関しては、従来多くの方式が提案されている[Gaede98]。高速と言われている最近の方式としては、SR-tree [Katayama97], VA-file[Weber98], A-tree[SakuraiOO]がある。

SR-tree は R*-tree[Beckmann90], SS-tree[White96]という B-tree の流れを汲む系列に属し、クラスタに分割する際、データ分布に依存してクラスタを決めるデータ分割の方式である。最近では、このデータ分割の方式が主流である。SR-tree ではクラスタとしては球と直方体の共通部分を用いているが、高次元での性能低下が指摘されている[Weber98, Katayama01]。データ分割とは対照的にクラスタを前もって決められた位置に基づいて分割する空間分割の方式としては quad-tree[Samet84]がよく知られている。

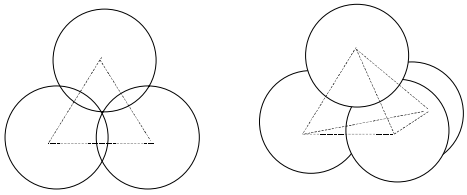
VA-file は直交座標を用いた近似に基づく方式で、通常の多次元インデクスとは異なり、階層構造を持たず、検索は逐次検索となるため、大規模データではその点が問題となる。

A-tree も直交座標による近似を用いているが、階層構造をもつ方式である。VA-file が近似で絶対座標を用いているのに対して、クラスタとしての直方体内の相対的な座標を用いている。SR-tree や VA-file よりも 64 次元で入出力回数を 1/4 程度に削減している。

直交座標による近似では、基本的に空間をセルと呼ばれる部分空間に分割し、そのアドレスを近似情報としている。

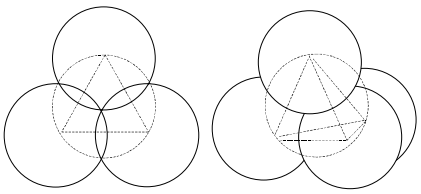
２．基本的な考え方

本章では、１章で述べた問題に対する我々のアプローチの基本的な考え方について述べる。



(a) 2次元の場合　　(b) 3次元の場合

図2.1 球による空間の被覆



(a) 2次元の場合　　(b) 3次元の場合

図2.2 球の基本的な分割

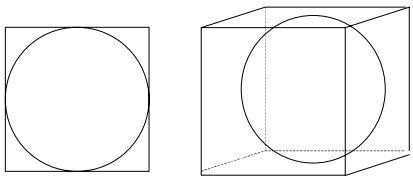
(1) 規則的球配置による高次元での高速化

図 2.1 のように、正三角形や正四面体の頂点に、重心から頂点までの距離を半径とする球（円も球と呼ぶことにする）を置くと、球の間に隙間なく、また重複も最小限にできる。球内の空間もその球と同じ半径の球で図 2.2 に示すように、隙間なく、また重複も最小限に分割することができる。ｎ次元で n+1 個の頂点を持ち、任意の頂点間の距離がみな等しい図形を正単体と呼ぶ。２次元、3 次元の正単体がそれぞれ正三角形、正四面体である。上記のことは n 次元の正単体でも成り立ち、n 次元の球は n+1 個の球で分割できる。

我々のアプローチは、このことを基本に一般には同じかより小さい半径の球で、空間を隙間なくまたなるべく重複なく、分割することである。[Katayama0]で述べられている SR-tree のクラスタ同士が高次元では大きく重なり合ってしまうという問題をこの方法で解消できる。しかし、この方法の問題点は、上述のようにデータ分割でなく空間分割であり、クラスタによっては含まれる点の数が少ないものができてしまうことである。すなわち、１つの球に１ページを割り当てると、スペース効率を落すことになる。このことと１章で述べた実現の容易性を考慮して、レコードベースでの実現を考えた。

(2) 極座標的な方向による近似

レコードベースの課題であるレコードへのアクセス回数を減らすために近似を用いる。球は SS-tree や SR-tree でも使われている。球内の空間を近似するために従来の直交座標による方法を用いると問題があることをまず示す。



(a) 2次元の場合　　(b) 3次元の場合

図2.3 球とその外接立方体

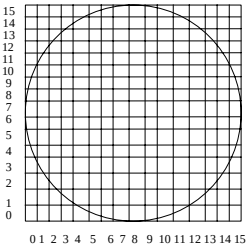


図2.4 球内の点の直交座標による近似

表2.1 立方体と球の体積比

次元	立方体の体積／球の体積
2	1.27
3	1.91
4	3.24
16	2.78×10^{5}
64	5.99×10^{38}
256	1.03×10^{229}

従来方式で近似しようとすると、図 2.3 に示すように球に外接する立方体内を直交座標で表現することになる。図 2.4 は２次元の場合にセルで空間を分割した様子を示すものである。この場合、球（２次元でも円ではなく、球と呼ぶことにする）の外側のセルは無駄となる。２次元の場合はこの無駄はまだ少ない。

表 2.1 は、各次元ごとの立方体のその内接球に対する体積比をまとめたものである。このように、高次元になると、無駄なセルが非常に多くなる。この無駄をなくせれば、近似情報をコンパクトにでき、類似検索の高速化が図れる。我々は、ある基準点から点への方向を方向ベクトルというもので近似することに基づく極座標に似たアプローチを採った。

(3) 再帰的次元分割と適合的ビット割り当て

(2)で述べた方向ベクトルを求める際に、基準点から点の方向への単位ベクトル（長さ１のベクトル）を均等な次元に再帰的に分割する。その際、分割された２つのベクトルに割り当てるビット数を長さが長い方により多く割り当てることが自然と思われる。現在、まだ試行錯誤で行なっている段階であるが、その１つの方法を 3.2 で説明する。

(4) 大規模データへの対応

VA-file のように､平坦な構造のインデクスが提案されている。検索は逐次処理となるが、データ量が少ない場合は十分である。また、インデクスを主記憶上にロードして高速化するという方式も考えられる。しかし、将来的には、データ量は急増し、これらの方式では扱えないニーズも増えると予想している。そこで、大規模なデータに対応するためには階層化は不可欠と考えた。

(5) 実現の容易性

データベースシステムの中核に手を入れずに実現できるのであれば、アプリケーションと同様、時間的にもコスト的にも開発は格段に容易になる。また、SQL や JDBC などに沿って実現すれば、 それに準拠した任意のデータベースシステムの上で稼働させることが可能になる。そこで、データベースシステムの上に実現するというアプローチを採った。

(6) 階層的な識別子

レコードベースでは、前述のように、クラスタリングを直接制御することはできない。しかし、間接的にデータベースシステムがクラスタリングをするように促すことは可能である。我々は、多次元インデクスの階層構造を反映した識別子を用いることにより、これを実現した。

３．方式

この章では、２で述べた基本的な考え方に基づく我々の方式についてより詳細に述べる。

３．１ 規則的球配置による多次元インデクス

２章で球内の空間が同じ半径の球で隙間なく分割できることを示した。次により一般的に球内の空間を半径が同じかより小さい球で覆う方法を示す。

基本的な考え方は、ルート球と呼ばれる全ての点を含む球から出発して、親の球（親球と呼ぶ）を同じかより小さい半径の球（子球と呼ぶ）に分割していき、子球をさらに再帰的に分割することにより階層的なインデクスを構成することである。１つの親球の子球の半径はみな同じであり、親球の半径に対する子球の半径の比を分割比と呼ぶ。1.0 の場合は同じであり、0.5 の場合は、子球の半径が親球の半分になる。

親球内を子球に分割する際は、球内に基準となる正単体を置き、親球に含まれる点がその頂点を中心とする球（これを頂点球と呼ぶ）に含まれれば、それに含め、もし含まれなければ、次に述べる拡張球を探す。基準正単

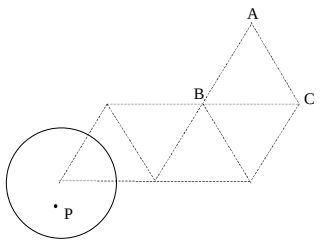


図3.1 正単体の接続

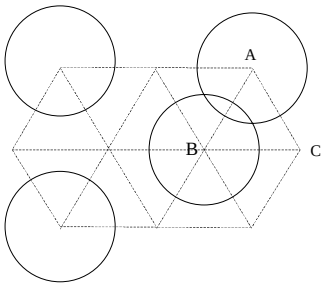


図3.2 拡張分割

体の大きさは、その重心から頂点までの距離が、子球の半径に等しくなるようにする。

図 3.1 はこのことを２次元で説明するものである。今、点 P を含む球を求めるものとする。点 P は基準正単体である正三角形 ABC のどの頂点球にも含まれていない。この場合、図に示すように点 P に向けて正単体を接続していくことにより、有限回の操作で正単体の頂点球が点 P を含むようにできる。この球を拡張球と呼ぶ。

以上をまとめて、球を再帰的に分割して、多次元インデクスを生成するアルゴリズムを以下に示す。

(1) 多次元インデクスの生成アルゴリズム

分割対象となっている球をS、Sに含まれる点の集合を、

$P(S) = \{p_1, p_2, \ldots, p_k\}$

とする。S を分割してできる子球の集合を C(S)とする。C(S)は最初は空集合である。子球は生成された順に、c_1, c_2, ... とする。また、S 内の適当な位置に置いた分割の基準となる正単体をσとする。基準正単体の位置については、この後に詳しく述べる。最初 S をルート球として、1)以下を行なう。

1) Sに含まれる点の個数が閾値 t 以下の場合は戻る。
2) P(S)に含まれる各点 p_i に対して以下を行なう。
3) p_i が C(S)に含まれる子球に含まれるかどうか c_1, c_2, ... の順に調べる。
   含まれる場合、その子球に含め，2) に戻る。
   含まれない場合、次の 4) を行なう。
4) σの頂点球に含まれるかどうか調べる。
   含まれる場合、その頂点球を生成し、点 p_i を含める。また、その頂点球を C(S)に加え、2)に戻る。
   含まれない場合は、次の 5)を行なう。
5) σから出発して正単体の拡張球を探し、点 p_i を含める。また、その頂点球を C(S)に加え、2)に戻る。
6) こうして求められた $C(S) = \{c_1, c_2, \ldots, c_j\}$ の各 c_i に対して、再帰的に 1)～5)を行なう。

図 3.2 はこうして分割された子球の例を示したものである。この図のように、必要な球だけ生成される。なお、閾値 t を分割閾値と呼ぶ。分割比とともにチューニングパラメータの一つである。

(2) 実質球

基準正単体の重心の位置を球内のどこに置くかということについては、次の２つが考えられる。

a) 球の中心
b) 球に含まれる点の重心

a) は、２章で説明したように頂点球を球と同じ半径にすると図 2.2 の基本的な分割に当たり、頂点球だけで球を分割できるという特徴がある。しかし、データに偏りがある場合、効果的な分割にならない可能性が高い。そこで、基本的には、b)が良いと考えており、実際のインプリメントでもこちらを採った。この点の重心を中心とし、これから最も遠い球内の点までの距離を半径とする球を考えると、検索時には、この球と交わるかどうかが実質的に問題になる。その意味で、この球を実質球と呼ぶ。

これに対し、規則的に配置された球を実質球と対比させる必要がある場合は規則球、通常は単に球と呼ぶ。

b)のアプローチでは、分割対象は実質球と考えられる。したがって、親の実質球の半径に対する子の規則球の半径の比を分割比とする。しかし、点の重心や重心から最も遠い点までの距離は点の追加、削除に伴い変動するため、このままでは子球を規則的に配置することはできない。このため、球の分割時以降は実質球の中心の位置は変えないようにする。この半径を特に分割時実質半径、動的に変わる方を動的実質半径または単に実質半径と呼ぶ。分割時実質半径は点の追加に伴い新しく球を生成する必要がある場合に、動的実質半径は検索時に用いられる。

以上のように我々の方式は基本的には空間分割であるが、実質球によりデータの分布をインデクス構成に反映させることができる。

なお、２次元では、図 3.2 に示すように、平面を正三角形で重複なくきれいに覆うことができる。しかし、３次元ではすでに、空間を正四面体で重複なくきれいに覆うことはできない。すなわち、正単体同士が交わってしまう場合が生じる。この意味で、「なるべく」重複なく球を配置するという言い方をしている。

３．２ 方向による近似

２章で、球内の点を直交座標で近似する際の問題点を述べた。ここでは、方向による極座標的なアプローチを具体的に述べる。

球内の点 P は実質球の中心を O、OP 方向の単位ベクトルを e, OP の長さを d とすると、(d, e) という組で表わせる。これは、極座標が原点からの距離と方向を表す角度で表されるのに対応している。このように近似の基準はVA-fileのように絶対座標によるものではなく、A-treeのように相対的である。

我々は、方向を量子化して、この e を次のように近似

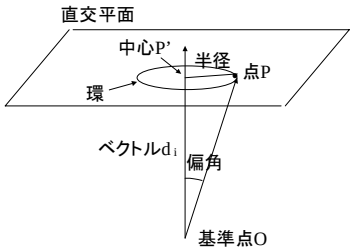


図3．3 環による近似

することを考えた。まず、方向ベクトルと呼ばれる単位ベクトルを複数個用意し、その集合を

$D = \{ d_1, d_2, \ldots, d_m \}$

とする。そして、ベクトル OP とのなす角度が最も小さい方向ベクトル d_i を e を近似するものとして、D から選ぶ。図３．３はこうして選ばれたベクトル d_i と点 P との関係を示している。ベクトル d_i に直交する点 P を含む平面と線分 OP またはその延長と交わる点を P'とする。すると、点 P は点 P'を中心とした半径 PP'の球面上にあることになる。さらに言えば、この球と直交平面とが交わってできる n-1 次元の球面（これを環と呼ぶ）上にある。環は、OP' = d, PP' = r とすると、

(i, d, r)

で表せるので、この３つ組で点 P を近似できる。i を方向ベクトルの識別子、d を軸長、r を環の半径と呼ぶ。また、P'を環の中心、OP と d_i のなす角を偏角と呼ぶ。なお、方向ベクトル d_i の識別子 i は一般には数バイトから数十バイトのビット列で表現される。図 3.4 はこうして各点に対応した環の様子を示したものである。ここで重要なことは、点 P から方向ベクトルの識別子 I を求めたり、逆に i から d_i を求めることが比較的容易に行えるように方向ベクトルの集合 D や各方向ベクトルの識別子を定める必要があるということである。

次に上記の中で特に問題となる方向の量子化、すなわち、方向ベクトルの集合 D と方向ベクトル識別子をどう設定するかについて述べる。

方向ベクトルはなるべく均等な方向を向いていることが望ましいと考えられる。３次元で考えると、正多面体の重心から各頂点、あるいは各面の重心への方向は明らかに均等な方向である。しかし、一般に m 個のベクトル

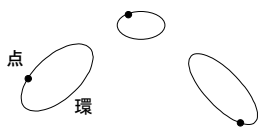


図3.4 各点に対応する環

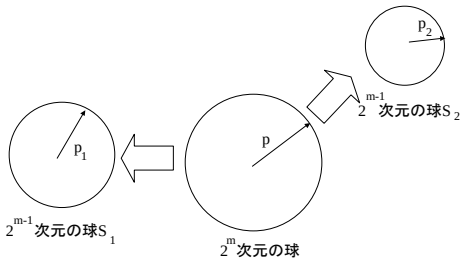


図3．５ 再帰的次元分割

となると、完全に均等な方向というのは無理で、なるべく均等な方向を見つけるということになるであろう。我々は、なるべく均等な方向に近付けるべく、いくつかの方法を試みた。その中で性能の良かった再帰的次元分割による方法とその評価の基準とするための直交座標による方法を以下に示す。

(1) 再帰的次元分割による方式

簡単のために、まず次元 n を 2^mとして説明する。

この方法では、近似しようとしている点に対応するベクトル OP と向きが同じ単位ベクトルを p とし、これを再帰的に２つの次元に分割していく中で、p との偏角が最も小さい方向ベクトルの識別子（以降、略して p の識別子と呼ぶ）を決めていく。図３．５はこの分割の様子を示したものである。

$p = (x_1, x_2, \ldots, x_n)$,

$p_1 = (x_1, x_2, \ldots, x_{n/2}, 0, 0, \ldots, 0)$

$p_2 = (0, \ldots, 0, x_{n/2+1}, x_{n/2+2}, \ldots, x_n)$

とすると、

$p = p_1 + p_2$

である。今、座標が 0 の部分を除いて、p_1, p_2 を改めて、

$p_1 = (x_1, x_2, \ldots, x_{n/2})$

$p_2 = (x_{n/2+1}, x_{n/2+2}, \ldots, x_n)$

とおく。p の終点は、半径|p|の 2^m次元の球上にあり、同様に、p_1, p_2 の終点はそれぞれ、半径| p_1|, | p_2|の 2^{m-1} 次元の球上にある。ここで、p の識別子を

γ | p_1の識別子| p_2の識別子

で表す。ここで｜はビット列の連結を意味する。γ は p_1 の p に対する長さの比

$\gamma = |p_1| / |p|$

である。

$|p|^2 = |p_1|^2 + |p_2|^2$

の関係があり、識別子から方向ベクトルを求める場合、|p|がわかっているので、| p_1|, | p_2|を計算するにはγだけで十分である。

ここで、p_1, p_2 の識別子にそれぞれ何ビットずつ割り当てるかが問題となる。今、γ に最初 k ビット割り当てるものとする。また、p_1 の識別子と p_2 の識別子に割り当てるビット数をそれぞれ b_1, b_2 とすると、

$b_1 = (n/2 - 1)k_1$,
$b_2 = (n/2 - 1)k_2$,
$k_1 + k_2 = 2k$

となるように割り振る。すなわち、p_1，p_2 全体で、2(n/2-1)k ビット割り当てるが、それを k_1 : k_2 の比で割り振ることを意味している。 p に割り当てられるビット数は(n-1)k ビットである(性能評価では、次の(2)の方式と合わせる意味で nk ビットとして扱った)。ここで、

b_1: b2 = $\log_2$ (球 S1 の表面積)：$\log_2$ (球 S2 の表面積)

が成り立つように、k_1 k_2 を定める。端数が出た場合は、k_1 k_2 の大きい方を切り上げ、小さい方を切り捨てる。

以上のことを再帰的に、p_1 ,p_2 に行うことにより、p の識別子を求めることができる。なお、p_1, p_2 の識別子のγには、それぞれ k_1、k_2 ビットを割り当てる。再帰的に処理を進めると、最後に、p_1，p_2 が２次元となる。この場合は、x 軸からの角度を量子化する。その際用いるビット数はそれぞれ k_1 または k_2 ビットである。

今まで、次元が 2^m の場合について説明してきた。一般の場合には、p の次元を n とした時、p を再帰的になるべく均等な次元に分割していく。すなわち、n = 2m の時は、２つの m 次元のベクトルに、n = 2m+1 の時は、m+1，m 次元に分割する。ビットの割り当てについても、式は少し複雑になるが、同様の考えで行う。なお、ｎが奇数の場合、最後に１次元になる場合が起きる。この時は、方向は正または負の２方向となり、１ビットで表現する。

こうして求めた識別子から点 P との偏角が最も小さい方向ベクトルは再帰的手続きにより容易に求まる。

(2)直交座標による方式

直交座標を量子化することにより、方向ベクトルを求める。今、近似する点を P とし、(1)と同様にベクトル OP と向きが同じ単位ベクトルの各座標を量子化する。各次元ごとに k ビット、n 次元では、全体で nk ビットを用いる。なお、量子化された情報から方向ベクトルを復元する場合は、対応するセルの重心に方向ベクトルが対応しているものとする。この方法は前に従来方式を説明した際と基本的に同じ考えであり、近似表現は冗長となる。

３.３格納方式

ここでは、商用としてよく使われている関係データベースシステムで点や索引のレコードを格納する方法について述べる。ただし、オブジェクト指向データベースで実現することも可能である。

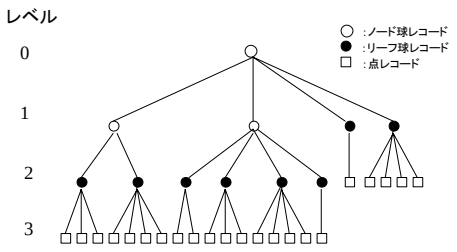


図3．６ 索引・点レコードの階層構造

我々の多次元インデクスは図 3.6 に示すように階層構造をとる。この階層構造を点リレーションと索引リレーションという２つのリレーションで実現する。

(1) 点リレーション

点に関する情報は点リレーションに格納する。１つのレコード（他との関係でタプルではなくこの用語を用いる）に１つの点の情報を格納する。点の情報としては、以下で述べる階層的識別子と、点の各次元ごとの座標値を要素とする配列をバイト列として格納する。階層的識別子にはインデクスを張る。

(2) 索引リレーション

各球に関する情報は、１つの球に１つの索引レコードと呼ぶレコードを対応させて索引リレーションに格納する。球の情報としては、階層的識別子と、球の種別、規則球の中心、実質球の中心、分割時実質半径、動的実質半径、球に含まれる子球または点の数、球に含まれる子球または点の情報を格納する。球の種別は、図 3.6 に示すノード球かリーフ球の違いを示す。球に含まれる子球または点の情報は、バイト列として格納する。子球の場合は、各子球の情報（階層的識別子と、近似情報としての実質球の中心に対応する環、実質半径）を要素とする配列、点の場合は、含まれる各点の情報（階層的識別子と近似情報としての点に対応する環）を要素とする配列である。階層的識別子にはインデクスを張る。

(3) 階層的識別子

上記の点や球の識別子は、(level, parentId, id)という３つ組からなる階層的な識別子とする。id は球や点にそれぞれ 1 から生成順に付けられるユニークな通し番号である。level は球または点の図 3.6 に示すレベルである。parentId は点や球の親球の id である。ルート球の場合は親球がないので、値は 0 とする。こうして定義した階層的識別子の辞書式順序に基づいて２つのリレーションをソートする。レコードは通常、挿入順に二次記憶上に格納されるので、ソートにより親球ごとにクラスタリングすることができる。ただし、点の追加や削除に対応するために、時々ソートによる再構成を行う必要がある。

通常サポートされているのは、逐次的なリレーションであるが、[Yamane89]にあるような B-tree 構造を持つリレーションをサポートすれば、再構成の必要はなくなる。

３.４ 類似検索

類似検索には、ある範囲内の点を検索する範囲検索と、ある指定された点から近い順に k 個の点を取り出す k-近傍検索がある。以下、我々の多次元インデクス方式を用

いたこれら２種類の検索方法を近似に関するポイントに絞って説明する。

(1) 範囲検索

ある指定された点からある半径の球（これを近傍と呼ぶ）内の点を求める処理である。基本的には、ルート球から階層をたどり、検索する。

ノード球の場合は、子球の実質球中心は環で近似されているため、球の存在範囲は、３次元で言えばトーラスとなる。このトーラスと近傍が交わるかを検査する。多次元でもその検査は容易である。

リーフ球の場合は、含まれる点を順に調べ、近似情報からその点が近傍に含まれる可能性があるか検査する。

(2) k-近傍検索

k-近傍検索において必要最小限のノードをたどるだけで解を見つけられることが知られており、その基本的な手法が [Katayama01]に記載されている。我々もこの手法を基本としている。以下、ここに記載されていない近似の部分について説明する。

[Katayama01]に書かれた手法では、ルート・ノードから出発し、近傍の中心から近い順に球をたどる。球までの正確な距離を計算しようとすると、索引レコードにアクセスしなければならない。我々の方式では、近似を使っているため、これでは近似の意味がなくなる。そこで、おおよその距離として、実質球の中心を近似している環の中心と近傍の中心間の距離を用いた。

なお、我々の方式では、索引レコードにアクセスした後、近傍が実質球と同時に規則球とも交わるかを判定して検索すべき球を絞ることが可能である。実質球が規則球に含まれるとは限らず、はみ出す可能性があるためである。実際の実現の際にも、この手法を用いた。

４． 性能評価

二次記憶を用いた試作を行う前に、主記憶上でのシミュレーション実験を行い、性能評価を行った。この章ではその結果について述べる。

４.１ SR-tree との性能比較

SR-tree との性能比較を点レコードや索引レコードへのアクセス回数に関して行った。SR-tree に関しては、フリーのソフト[*1]を用いてアクセス回数を求めた。

用いた特徴量データは、16,763 件の画像データから色の 64 次元の特徴量を抽出したものである。その中から 31 件の k-近傍検索を行った結果を表 4.1 に示す。近似のための環の情報としては、方向ベクトルのために32バイト、軸長と環の半径を合わせて合計40バイトを用いた。軸長、半径についても量子化することは可能であるが、今回は行っていない。分割閾値は 200 とした。

合計のレコードアクセス回数では、分割比が 0.5 の場合 1/6 程度に，分割比が 1.0 の場合 1/4.6 程度に SR-tree のレコード回数を削減している。ただし、以下の点がこの結果には反映されていない。

1) 我々の方式ではクラスタリングを間接的にしか制御できない
2) 階層的識別子に張られる B-tree インデクス
3) 近似によるオーバヘッド
4) ノード球に対応する索引レコードの構成の違い

したがって、レコードアクセス回数での比較がそのまま性能の違いを意味するものではない。2)については影響する可能性があるが、それ以外の影響はそれほど大きくはないと考えている。

まず、1)については、階層的識別子によって、かなり緩和できると考えている。2)に関しては、2〜5 割程度のオーバヘッドはあるものと考えている。B-tree による検索は、特に検索対象の二次記憶上での分布に大きく影響されるが、1)で述べたことからこの影響は抑えられると考えている。3)のオーバヘッドは、近似情報が 40 バイトと、点の座標データ 256 バイト（浮動小数点 4 バイト×64 次元）の 1/6 以下であり、アクセスされる近似情報は全体の半分程度であるため、全体としてのオーバヘッドは１割に満たないと考えている。4)に関しては、SR-tree の索引レコードがその球の枝刈りに関する情報は含んでいるが、子に関する情報は、その索引レコードとは別に管理しているのに対して、我々の方式では、両方索引レコードで管理している点が異なる。すなわち、我々の方が索引レコードにアクセスするとオーバヘッドが大きい可能性がある。ただし、表４.1 において示されるように、ノード球の索引レコードへのアクセス回数は少ないため、この影響は少ないと考えている。

インデクスの生成時間に関しては、分割比が 1.0 の場合は 17 秒であったが、0.5 の場合は約 8 分 30 秒を要した。この点も特に大規模データになった場合に課題である。

表4.1 SR-treeとの比較

方式	分割比	アクセス回数		
		点レコード	索引レコード	合計
我々の方式	0.5	136	364(23.4)	500
	1.0	200	457(29.7)	657
SR-tree		1988	1030	3018

注: 括弧内はノード球の索引レコードへのアクセス回数

[*1] http:// research.nii.ac.jp/~katayama /homepage/research/srtree/Japanese.html

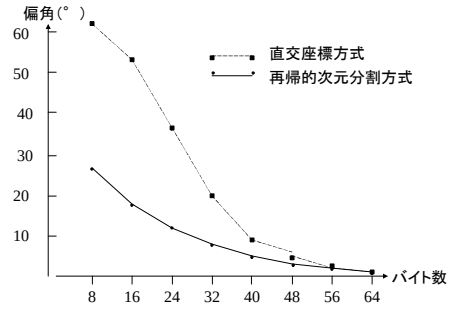


図4．1　識別子のバイト数と偏角の関係

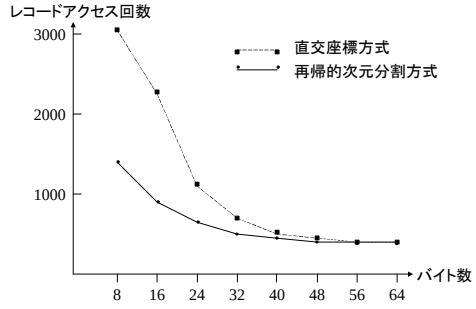


図4．2　レコードアクセス回数との関係

４．２ 近似の手法の比較

3.2 で説明した方向ベクトルを量子化するための２つの方式を比較した。用いたデータや検索内容は、4.1 と全く同様である。なお、分割比は 0.5 とした。図４．１は２つの方式の方向ベクトルの識別子のバイト数と点や球を近似する全ての環における平均偏角の関係を、図４．２は同じく近似情報のバイト数と合計アクセス回数の関係を示す。

両図とも再帰的次元分割方式が直交座標方式より優っていることを示している。特に、方向ベクトルの識別子が短いほど優るという傾向を示している。

５．まとめ

レコードアクセス回数に関する性能比較では、SR-tree1 の 1/6 程度に削減しており、良好な結果と考えている。ただし、4.1 で述べたオーバヘッドの影響も含めるべく、今後、二次記憶による実現を行い、さらに評価を進める予定である。

方向ベクトルによる近似に関しては、再帰的次元分割の方法が直交座標による方法に比べ、特に長さが短い場合によい結果を出している。球内空間の近似方法の評価は、立方体の体積とそれに内接する球の体積の関係からより理論的にできるはずである。今後、この観点からも評価や改良を行いたいと考えている。

謝辞

本研究に関しては、富士通研究所の長田茂美氏、増本大器氏、上原祐介氏、遠藤進氏、富士通大分ソフトウェアラボラトリ下村照雄氏、安藤淳禎氏に協力していただいた。ここに謝意を表したい。

【参考文献】

[Beckmann90] N. Beckmann:
The R*-tree: An efficient and robust access method for points and rectangles,
Proc. SIGMOD 1990, pp.322-331 (1990).

[Carey96] M.J.Carey, et al.:
Of Objects and Databases: A Decade of Turmoil,
Proc. 22nd VLDB,(1996).

[ChaudhuriOO] S. Chaudhuri et al.:
Rethinking Database System Architecture: Towards a Self-tuning RISC-style Database System,
Proc. 26th VLDB,(2000).

[Gaede98] V. Gaede et al.:
Multidimensional Access Methods,
ACM Computing Surveys, Vol. 30, No.2, (June 1998).

[Katayama97] N. Katayama, et al:
The SR-tree: An Index Structure for High-Dimensional Nearest Neighbor Queries,
Proc. SIGMOD 1997, pp.369-380 (1997).

[Katayama01] 片山紀生 他:
類似検索のための索引技術、
情報処理 Vol.42 No.10, pp.958-964, (October 2001).

[SakuraiOO] Y. Sakurai, et al.:
The A-tree: An Index Structure for High-Dimensional Spaces Using Relative Approximation,
Proc. 26th VLDB, pp.516-526 (2000).

[Samet84] H. Samet:
The quadtree and related hierarchical data structure,
ACM Computing Surveys, Vol.16, No.2, 187-260, (1984).

[Weber98] R. Weber et al.:
A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces,
Proc. 24th VLDB, pp.194-205 (1998).

[White96] D. A. White, et al.:
Similarity Indexing with the SS-tree,
Proc. 12th ICDE, pp.516-523 (1996).

[Yamane89] Y.Yamane, et al.:
Design and Evaluation of a High-Speed Extended Relational Database Engine, XRDB,
Proc. 1st DASFAA, pp.52-60, (1989).