時空間データベースインデックス正規化 R*-tree の実装と性能テスト

堀之口 浩征† 黒木 進† 牧之内 顕文†

時空間データベースは時間と空間の2つの異なる属性を合わせ持つデータを管理するデータベースである．時空間データベースに対する時空間検索を高速化する目的で用意される時空間インデックスの設計では，この両属性の性質の違いを考慮する必要がある．時空間データの属性値は幅のある多次元データと考えられる．そこで，我々は空間インデックスとして有名な R*-tree を時空間領域へ適用することにした．しかし，単に時間軸を R*-tree に追加しただけでは，時間属性と空間属性の異なる性質から不適切なクラスタリングを行う可能性がある．本論文ではこの問題を回避する手段を提案する．また，今回提案するインデックス構造がどのような性質の問合せについて有効であるかを計算機実験により検証した．結果として，より有効な時空間インデックスに適したデータ構造を提案する．

Normalized R*-tree for Spatiotemporal Database and Its Performance Tests

HIROYUKI HORINOKUCHI,† SUSUMU KUROKI† and AKIFUMI MAKINOUCHI†

The spatiotemporal index takes very important role on accelerating the spatiotemporal query processing. Spatiotemporal database have two distinct attributes simultaneously, so spatiotemporal index must combine those two irrelevant values and build structures for later query. We use R*-tree based structure for spatiotemporal index. The R*-tree was invented mainly for spatial area, and does not fit into spatiotemporal space without losing some its advantaged effeciency under certain situation. At first we point out the problem of R*-tree on spatiotemporal space and other inadequate environments. Then, we propose new technique to operate on that problem and show its performance with simulations.

1. まえがき

時空間データベースでは，時間属性と空間属性を持つデータを管理する．たとえば，物体の移動の記述として，空間位置の経時変化をとる方法が考えられる．このデータを保持する時空間データベースには，任意の時刻における位置や，特定の領域を通過する時刻等の問合せがなされる．

時間変化を持つデータとしては，従業員の給与が変化していく過程を考えることもできる．このデータでは空間的な属性を持ち合わせていない．しかし，データの給与の属性値は空間属性値と同じ数値で表現されるので，1つの空間属性値と見なせば時空間データベースで扱うことも可能となる．

† 九州大学大学院システム情報科学研究科
Department of Intelligent Systems, Graduate School of Information Science and Electrical Engineering, Kyushu University

これらの例にあるように，時間属性と空間属性の位相的な側面に着目することで，時空間データを統一的に扱うことが可能となる．我々の研究室では，時空間データを位相的に扱える時空間データベースとして，Hawks[4] と名付けたシステムの開発を現在進めている．Hawks では，対象を高次元の図形と見なし，位相空間の中で統一的に管理するための位相空間データモデル Universe[16] を提案している．今回の時空間インデックスは，この Hawks での検索効率向上を目的として研究開発を進めている．

我々は，時空間インデックス実現の足掛かりとして，比較的十分な研究成果が得られている空間インデックスを採用した．しかし，空間インデックスでは空間の属性においてすべて共通した単位を用いるとの仮定が存在する．時空間インデックスではこの仮定を取り除き，時間属性と空間属性といった異なる性質を持つ属性を統一的に扱う必要がある．

そこで，今回提案する手法では正規化[4] と呼ぶ変換

を木の構築時に導入し，木構造の改良を試みた．結果として，時空間領域の特徴に応じた本検索手法の有効性を確認した．

本論文の構成は以下のとおりである．まず，2 章において，時空間インデックスとして何が必要であるか，従来の手法での問題点は何かを論じる．3 章では，従来の空間検索の手法を単純に時空間領域へ応用した場合の問題点を具体的な状況を例にとって指摘する．そのうえで，我々の導入した時空間領域への適応の方法を紹介する．4 章は，計算機実験により実際の効果を調べ評価する．5 章では，関連する研究について説明し，最後に 6 章で今回のインデックスについてまとめる．

2. 時空間インデックス

まず，時空間インデックス構造の足掛かりとした空間インデックス，特にその中心となる構造の R*-tree[9] について説明する．

2.1 空間インデックス

空間データベースの応用例としては，NASA の EOS-DIS プロジェクトがあげられる．同プロジェクトでは，地球科学のための膨大なデータを空間データベースにより管理し，このデータを効率的に利用する手法を研究している．Sequoia 2000 project[8] のベンチマークもその 1 つで，川や湖等のオブジェクトの位置を数値化し，その形状を Point, Raster, Polygon, Graph で表現している．この数種の形式のデータを効率的に管理し，問合せを効率良く処理するために用いられるのが空間インデックスである．

空間インデックスでは，内部のデータ構造の操作を容易にするために，対象のオブジェクトを近似形状で代用する．MBR（Minimum Bounding Rectangle）は，これらの近似形状の 1 つであり，対象オブジェクト（群）を包含する最小の矩形でかつ各辺が対応する空間軸に平行な領域である．具体的には，二次元の MBR は長方形であり，三次元では直方体となる．

MBR の二次元の例として，**図 1** をあげる．ここで，斜線領域で表されるオブジェクトの MBR は，オブジェクトを囲む太い線の矩形である．そして，この MBR は P_1 (X1,Y1) と P_2 (X2,Y2) の 2 点で表される．もし対象が 3 次元のオブジェクトならば，3 次元の一対の点 P_1 (X1,Y1,Z1) および P_2 (X2,Y2,Z2) の形で表せる．同様にして，多数の点集合として表現される Polygon や Graph 等のオブジェクトも MBR により 2 点のみで近似できる．MBR の利用により，オブジェクトの位置管理の負担を小さくできる．

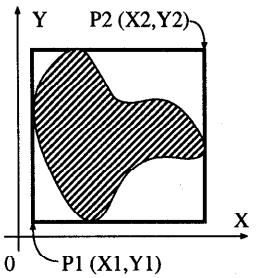


図 1 MBR の例
Fig. 1 An example of MBR.

空間インデックスを大まかに分類すると，n 次元 MBR を直接に用いる方法とそれを 2n 次元の領域等に変換して扱う方法の 2 つがあげられる[7]．たとえば，上記の 2 次元の P_1 と P_2 とで表される MBR は 4 次元の点 (X1, Y1, X2, Y2) で表すことが可能である．MBR 間には交差領域が存在するが，点間には存在しない．そのために点集合には MBR 集合に比べ，より単純にクラスタリングができる．一方で，空間インデックスにおいてよくなされる問合せの 1 つである最近接検索では，直接 MBR を用いる方が優れているとの結果も出ている[10]．我々は，直接 MBR を用いる方法を選び，その中で広い応用範囲で良い性能を示している R*-tree を使う．

2.1.1 R*-tree の構成

空間インデックスでは幅を持つデータを対象とする．幅を持つ物体群を対象にしたインデックスとして R-tree[1] が有名であり，R*-tree をはじめ，いくつか類似の構造[9),13),15)] を派生させている．

R*-tree や R-tree では，すべてのオブジェクトの MBR を多分木の構造で管理する．各ノードは 1 つの MBR と複数の子ノードへのポインタを持つ．葉ノードに限っては子ノードを持たないが，代わりにデータベース内のオブジェクト自身（のレコード）への参照を持つ．また，葉ノードが持つ MBR は，対応するオブジェクトの MBR であり，中間ノードには自分の各子ノードの MBR の集合を包含する MBR がある．つまり，ノードの親子関係が，MBR 間の包含関係と対応している．木は完全平衡木であり，葉ノードのみがオブジェクトへの参照を持つ．

R*-tree ではオブジェクトの投入により木が構築される．対象のオブジェクトがどのノードの下に付くかは，オブジェクトの MBR と中間ノードの MBR との位置関係により決まる．また，1 つのノードが持つことのできる子ノード数には上限と下限がある．上限に達したノードに新しくノードを追加しようとすると

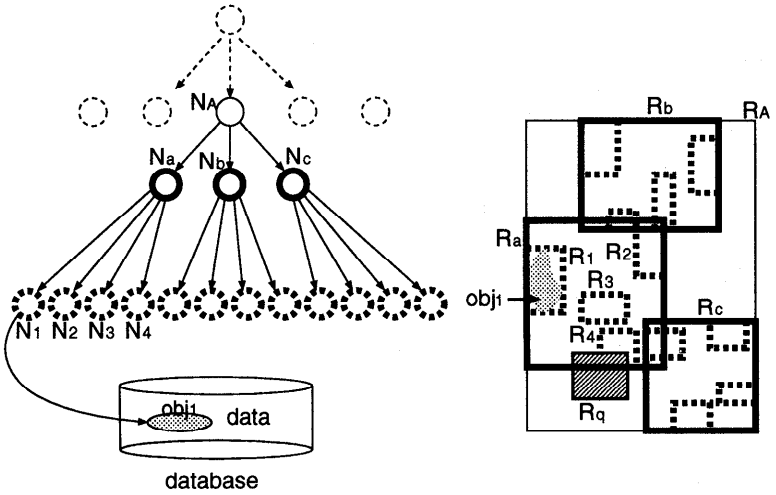


図 2 R*-tree の構成
Fig. 2 Structure of R*-tree.

きには，そのノードは分割手続き *Split* により分割される．ノード数の上限に最初に達するのは葉ノードの親のノードである．この親ノードが上限に達すれば，その親ノードを分割するように，この分割は葉ノードから根ノードに向けて伝播していく．結果として，葉ノードはすべて同じ高さに存在する．

通常 R*-tree を用いた領域検索（range query）には，'矩形'（すなわち，2 次元では長方形，3 次元では直方体）が指示される．あるノードの MBR が与えられた矩形と交わるならば，その子ノードの中に矩形と重なる MBR を持つものが存在する可能性がある．そこで領域検索では，与えられた矩形と重なる MBR を持つノードを根から葉へ向けてたどる．R*-tree を用いた領域検索の出力は，与えられた MBR と交差を持つ葉ノードが指すオブジェクトの集合である．

図 2 は二次元空間のオブジェクトに対する R*-tree の構成の例である．図の左側には R*-tree が，また右側には MBR の包含関係が描かれている．たとえば，葉ノード N_1 に対応する MBR が R_1 であるように，同じ線種および同じ添字を用いているノードと MBR を対応させている．同様に，葉ノード N_1 がデータベース中のオブジェクト obj_1 への参照を持つのに対応させて，右図の R_1 を obj_1 の MBR としている．左図のノード N_a に対する子ノード N_1，N_2，N_3，および N_4 の親子関係は，右図における R_a と MBR の集合 R_1，R_2，R_3，R_4 との包含関係として表れている．この木に対して，矩形領域（図 2 の右図の）R_q の領域検索を行う場合には，根ノードから葉ノードに向けて次の操作を行う．今 N_A に至ったとすると，N_A の MBR（つまり R_A）と R_q との交差を調べる．もし交差があれば，N_A の子ノードの N_a，N_b，N_c を検索の対象とし，その各 MBR の R_a，R_b，R_c と R_q の領域の交差を調べる．このときには，R_a のみが該当するので，同様の作業を R_a の子ノードに繰り返し，結果として R_4 を得る．R_4 は葉ノードであるので，R_4 がこの領域検索の出力となる．

以上は R-tree と R*-tree に共通した構成であるが，R*-tree では R-tree に加えて，実験で有効であると確認された手法を用いて，検索効率を向上させている．R-tree では中間ノードの持つ MBR の Area（二次元空間の場合では長方形の**面積**，三次元の場合には直方体の**体積**）をノード選択時の比較対象とする．一方，R*-tree では，必ずしも Area のみを比較対象とするのではなく，Margin（MBR の各辺の合計）や Length（MBR の中心点間の**距離**）等の基準を木の構築時の各所で利用する．軸の属性（単位等）の異なる位相空間（たとえば，時空間領域）に対して R*-tree を構築する場合には，このような比較基準を用いては問題が出ることは容易に推測できる．詳しくは 3 章において述べる．

2.2 Hawks の構成

R*-tree を用いた条件探索では，MBR のような近似形状を用いるため，その出力は，データベースの領域検索の処理結果としては不正確である．つまり，MBR のみを用いているので，出力集合にはすべての答に加えて答ではないものも含まれる．そのため，出力集合中の各オブジェクトについて，実際の時空間形状（MBR とは異なる）を調べ，問合せの条件を正確

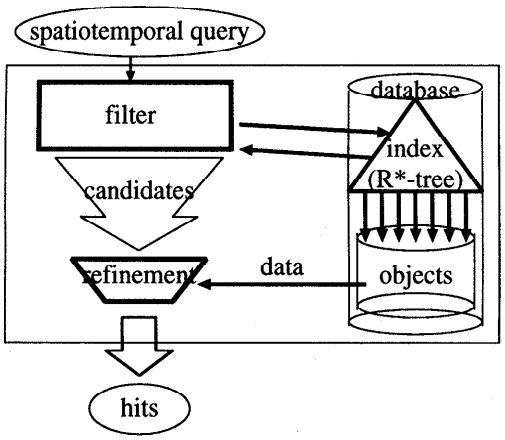


図 3 問合せ処理の流れ
Fig. 3 Flow of query processing.

に満たすかどうかを検査しなければならない．

Hawks における時空間インデックスは，近似に基づく質問処理[14] に基づいて構成され，図 3 に示すような構造を持つ．R*-tree を用いる部分は，図 3 のフィルタ（filter）である．フィルタの出力を受けるリファインメント（refinement）では，データベースからオブジェクトを取得したうえで，それが持つ正確な時空間属性値が問合せの（時空間に関する）条件を満たすか否かを判断する．オブジェクトの取得には大量の disk I/O が必要となるために，リファインメントは一般に重い処理となる．そのため，フィルタで条件を満足しないオブジェクトを可能な限り排除することが重要となる．

3. R*-tree の時空間データへの適用

時空間オブジェクトを R*-tree で管理するならば，投入しようとする時空間オブジェクトの MBR と，投入先の候補となるノードの MBR とをいくつかの基準で比較しなければならない．この‘比較’を行うときに，軸が表す属性（時間属性か空間属性，また，それらの単位等）を無視して比較することも可能である．しかし，属性が異なる値ではとりうる値の範囲も異なり，単純な比較では不適切にクラスタリングされる状況が生じうる．不適切にクラスタリングされる状況を述べる前に，まず R*-tree の構成を説明する．

3.1 R*-tree の構成

R*-tree の各ノードは *MBR* と複数の子ノードへのポインタを要素として持つ．また，1 つのオブジェクトに 1 つの葉ノードが対応する．R*-tree の構築は，オブジェクトから MBR を計算し，それを要素とするノードを木に投入することでなされる．そこで，まず

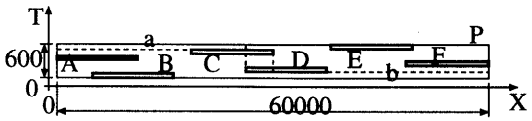


図 4 問題の生じる中間ノードの例
Fig. 4 Using values in an intermediate node.

投入の手続き InsertData の概略を次に示す．手続き ReInsert，ChooseSplitAxis，ChooseSplitIndex については，必要になった時点で説明する．

- 手続き InsertData
  (1) 子ノードとして追加すべきノードを選択する
  (2) その子ノード数がすでに上限値であれば，手続き OverFlowTreatment を実行する．
  (3) 手続き OverFlowTreatment の結果として当該ノードに手続き Split が実行された場合は当該ノードの親ノードに手続き OverFlowTreatment を実行する．
  (4) 親が根ノードならば，新しく親を作り，それを根ノードとする．
  (5) MBR を修正する．
- 手続き OverFlowTreatment
  (1) 自身が根ノード以外のノード（つまり，中間ノードもしくは葉ノード）であり，その高さで最初に手続き ReInsert から呼ばれたノードである場合には手続き ReInsert を実行する．そうでなければ手続き Split を実行する．
- 手続き Split
  (1) 手続き ChooseSplitAxis により，どの軸に垂直な直線で子ノードの MBR 群を分割するかを決める．
  (2) 手続き ChooseSplitIndex により，子ノードの MBR 群を分割する位置を決める．
  (3) 子ノードを 2 つのノードに分配する．

3.2 問題となる比較尺度

先に述べたとおり，R*-tree の構築過程の各手続きにおいては，Area，Margin および，Length の 3 つの尺度が用いられる．これらの尺度は R*-tree を時空間に適用するときに問題となる可能性を持つ．問題となる状況の二次元での例として図 4 を用いる．図 4 では，P で表される MBR を持つノードがあり，その子ノードの MBR が A, B, C, D, E, F とする．縦軸 T は月単位で表される時間軸，横軸 X は m（メートル）で表される空間軸であるとし，MBR P は，X 軸方向の長さが 600，T 軸方向の長さが 60000 であるとする．これは，50 年間に距離 60 km の範囲にある何かの経時変化を記録するとして，あながち不適当とは

いえない状況を表現している．図では縦と横で単位が異なるため，そのため軸の数値で表される形状（図では長方形）の各辺の長さに釣合いがとれていない状況である．すなわち，MBRは潰れた形になる．

3.2.1 手続き ChooseSplitAxis と Margin

手続き ChooseSplitAxis は手続き Split から呼ばれる．手続き Split は新しく自分の兄弟ノードを1つ作り，そこへ自分の子ノードのいくつかをつなぎ替える．手続き ChooseSplitAxis はこの移動するノード群を決定する場面で用いられる．

まず，子ノードをその MBR の座標値の最小値により昇順に整列する．最小値の同じ MBR についてはさらに最大値を用いてならべる．たとえば，図4を X 軸で整列すると，MBR の A，B，C，D，E，F の順となる．

ノードの持てる子ノード数には上限と下限があり，それをそれぞれ max と min とする．先頭からのいくつかとその残りという分け方をすると，子ノードの分割には $(\max - 2\min + 2)$ 通りがそれぞれ軸について存在する．分割させられた2つのノードの MBR の集合のそれぞれに MBR を決めることができ，この2つの MBR の Margin の和を Margin-value とする．この分割とそれにより作られる2つの MBR の例は図4での破線の矩形 'a' と 'b' として見ることできる．手続き Split の分割の軸は，$(M - 2m + 2)$ 通りの Margin-value の総和をすべての軸について計算し，最小の総和を持つ軸である．

R*-tree で Margin が用いられるのは，"中間ノードの持つ MBR を小さくできるからである" と説明される[9]．小さい Margin を持つほど MBR が（二次元の場合に）正方形に近付き，MBR が正方形に近付くほど中間ノードの MBR が小さくなる．しかし，分割対象のノードと子ノードのそれぞれの MBR の形に偏りがある場合には，特定の軸のみが選択される．図4の状況では横軸のみが選ばれるのである．結果として，縦軸についての情報はクラスタリングに利用されない．元々，R*-tree は空間インデックスであり，同一属性（しかも同一の単位）のみの数値が比較されると仮定できた．しかし，時空間では時間と空間の異なる属性が存在する．単位の異なる数値から得られた Margin を利用するのは無理がある．

3.2.2 手続き ChooseSplitIndex と Area

手続き ChooseSplitIndex では，まず手続き ChooseSplitAxis と同じ整列を行う．そして，それぞれの分割の中から2つの MBR 間の交差領域が持つ Area が最も小さな分割法を選択する．この Area も手続き ChooseSplitIndex の Margin を使う場合と同様，図4の例の場合では，横軸を狭くとる分割を多く選ぶ偏りが生じる．

この R*-tree の単純な利用では，X 軸の 600（MBR P の X 軸の 1/100）と T 軸の最大値である 600 とを同列に比較することになる．

3.2.3 手続き ReInsert と Length

手続き ReInsert は上限値に達したノードの子ノードを Length により整列する．そのうえで，Length の大きな子ノードのいくつかをノードから外し，再び木に投入する．外されたノードは元と同じ高さにつなげる．同じ高さのノードで再びノード数が上限値に達すれば，今度は手続き OverFlowTreatment に従い，分割（手続 Split）が実行される．この Length についても，他の比較基準と同様に，基点となる点の位置からの横軸成分を大きく持つ方が遠いと認識され，優先的に再投入される．

3.3 時空間領域の正規化

図4の状況では，横軸方向の分散が「クラスタリングに影響を与えない」という弊害が生じていた．この問題の原因は，もともと比較できない単位を持つ数値をそのまま比べていたところにある．異なる属性を持つ空間に R*-tree を適用する正規化の一手段として，与えられた数値をノード単位に変換する動的手法を我々は提案する．この手法では比較できない属性軸を持つ R*-tree の構築時に，根ノード以外のノードが持つ MBR の各次元の座標値を [0, 1] の空間内に再配置する．これにより各軸についての親ノードの MBR と子ノードの MBR の対応する辺の比を保つ．具体的には，自分の子ノードを [0, 1] の空間内に収めるために，各子ノードの MBR のそれぞれの辺の長さを自分の MBR の対応する辺の長さで割る．正規化を行う R*-tree では，正規化を施した数値から Area，Margin，Length を計算する．正規化により各属性は単位を失い，数値の相互比較が可能となる．以後，正規化を行う R*-tree を NR*-tree（Normalized R*-tree）と表記する．図4を例にとると，正規化により各 MBR は**図5**にある単位を持たない [0,1] に再配置される．

我々の方法は，オブジェクトの投入時において，それが投入されるべき部分木の直接の親ノードの MBR とそのオブジェクトの MBR との比較を行う動的な手法である．他の動的な正規化手法として，オブジェクトの投入時の根ノードの MBR と個々のノードの MBR を比べる手法も考えられる．根ノードの MBR とは，木で管理されている全オブジェクトの分布している空間と一致するために，個々のオブジェクトの MBR と

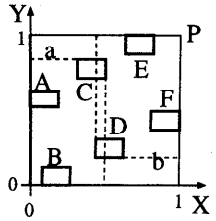


図 5 図 4 に正規化を行った場合
Fig. 5 Normalized on Fig. 4.

それらの分布する空間とに相似に近い関係がある場合には意味を持つと考えられる．ただ，この仮定は一般性を持つとはいえないので，汎用性を欠く問題点が残る．

これらの動的手法の対極として，静的に決定した領域を比較基準に用いる手法も考えられる．オブジェクトの比較基準の候補として次にあげる領域が考えられる．

- 定義域に基づく正規化
  ここでの定義域とは，対象のオブジェクトが分布する空間を指している．
- オブジェクトの大きさに基づく正規化
  オブジェクトの平均の大きさを基準にする手法である．
- 質問領域の大きさに基づく正規化
  オブジェクトの領域を正規化するために質問領域を用いる．

以上にあげた各手法は，オブジェクトの分布範囲や大きさ等があらかじめ判明しているか，もしくは予想できなければならないためにインデックスの対象が限定される．我々は，データベースに動的に格納されるオブジェクトを対象として考えており，初期にすべてのオブジェクトを一度に格納できる状況は仮定していない．したがって，我々は動的なクラスタリングが可能である R*-tree をインデックスとして採用し，そのうえで '動的な' 正規化手法を用意した．

4. 評　　価

NR*-tree と R*-tree との検索効率を比較するために，それぞれを実装し，ベンチマーク用データベースを使って実験を行った．

4.1 NR*-tree の構造とその構成

NR*-tree を実装するにあたり，我々の研究室で開発中のオブジェクトデータベースシステム出世魚とそのうえのデータベースプログラミング言語 INADA/ODMG を用いた．INADA/ODMG は ODMG[11),12)] で提唱されている言語仕様の C++バインディングを満たすライブラリで，出世魚が提供する永続オブジェクト処理のための手続きを実現する．

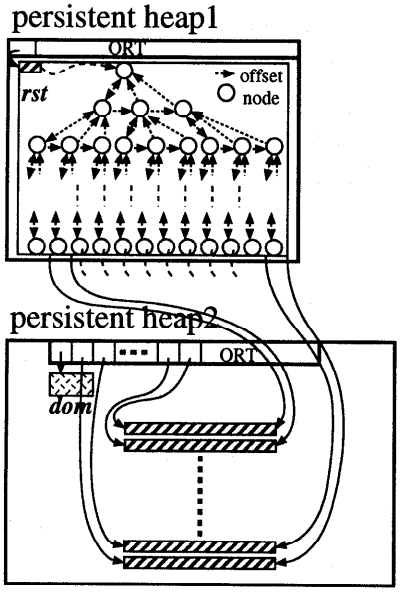


図 6 実験で用いた永続ヒープの構成
Fig. 6 Component of persistent heaps.

実装したベンチマーク用オブジェクトデータベースの全体的な構成は図 6 のとおりで，2 つの永続ヒープからなる．永続ヒープは仮想メモリ空間内の連続領域であり，OS のメモリマップドファイルの機能によりディスク上のファイルにマップされている．永続ヒープ上に作成されるオブジェクトは，それがマップされているファイル上に格納されることで永続性を得る．仮想メモリ上のオブジェクトへの操作はファイル中の対応するオブジェクトにも反映され，まったく同じ構造を持つ．今回のベンチマーク用オブジェクトデータベースでは 2 つの永続ヒープ *persistent heap1* と *persistent heap2* を用いた．

persistent heap1 は (N)R*-tree を実現するクラス d_RSTree のインスタンスである *rst* を格納する．rst を操作するプログラムは，実行時のオプションにより R*-tree か NR*-tree のどちらになるかを切り換えられる．rst は木全体を管理し，親ノードへの永続ポインタや子ノード数の上限および下限を保持する．今回の実験では，子ノードの数の上限と下限として**表 1** の値を用いた．

rst への入力であるオブジェクト群は persistent heap2 に格納している．このオブジェクト群は INADA/ODMG の d_Bag クラスのインスタンスである dom により管理される．

表1 子ノードの数の上限と下限
Table 1 Upper and lower limits of node entries.

	最大値	最小値
R*-tree	25	8
NR*-tree	25	8

表2 実験環境
Table 2 Enviroment of experiment.

maker	Sun Microsystems, Inc.
OS	SunOS 5.5.1
CPU	UltraSPARC 167MHz
platform	SUNW,Ultra-1
memory size	65536KB

永続ヒープはその中のオブジェクトを管理するテーブルとして，ORT（Object Reference Table）を持つ．persistent heap1 の rst や persistent heap2 の個々のオブジェクトもそれぞれの ORT により管理される．

また実験には，**表2** に示した性能のワークステーションを用いた．

4.2 実験内容

本実験では，任意の与領域内に存在するオブジェクトを検索するという領域検索を実行する．今回の実験ではインデックスによるフィルタの性能を調べるのであり，オブジェクトの形状として Polygon や Graph 等の複雑な形状のオブジェクトを作る必要はない．そこで，オブジェクトの MBR でオブジェクトの形状を代用する．今回は三次元（時）空間を対象とし，オブジェクトとして三次元の矩形領域（すなわち，直方体）を用いた．性能の変化を見やすくするために，あらかじめ定めた範囲内に一様分布させた特定の大きさのオブジェクトを 100,000 個用意する．そのうえで，このオブジェクト群に対して R*-tree と NR*-tree の各々でインデックスを作る．

オブジェクトと同じ理由で，領域検索で与える領域にも直方体を用いる．固定した大きさでランダムな位置にある複数の領域を用意し，領域検索を実行する．実験では，各領域検索の実行過程でたどった (N)R*-tree 中のノード数を数える．この処理をオブジェクトの 20,000 個を投入するごとに 25 回実行する．極端な値を持つ結果が全体に与える影響を小さくするために，25 回の実行結果中からその 1 割にあたる 3 回分の結果を最小と最大の側からそれぞれ除く．インデックスの性能としては，検索時にたどったノード数を用いる．ノード数を数え上げるのは，検索に実際に必要となる時間が検索時にたどったノード数にほぼ比例すると考えられるからである．投入されるオブジェクトが 100,000 個になるまで行われた計 110 回分（$(25-3)\times 5$）の領域検索の平均により，R*-tree と NR*-tree を比較する．

R*-tree と NR*-tree の性能比較に，次式で示される評価値を用いる．

$$\text{評価値} = \frac{\text{NR*-tree で取得された MBR 数}}{\text{R*-tree で取得された MBR 数}}$$

つまり，この値が '1' であれば NR*-tree は R*-tree と同程度の処理効率であり，'0.7' であれば正規化した方が非正規化の場合に比べて 3 割の効率改善を得られることを意味する．

4.3 実験結果と考察

今回の正規化の対象としているのは，オブジェクトの存在する三次元空間（以下定義域と呼ぶ）とオブジェクト（の MBR）の各辺の数値上の長さが軸ごとで大きく異なる偏った状況である．

そこで実験では，オブジェクトの MBR の各辺の長さを変えながら R*-tree と NR*-tree の性能比較をする．まず，定義域を固定し，直方体の各辺の比率を操作して性能の変化を見る．定義域が固定であるので，ある軸に関して直方体が他の辺に比べ長い辺を持つならば，その軸では物体間の交差する割合が増す．つまり，ここでは物体間の辺の混み具合の性能に与える影響を調べる．

次に，最初の実験から得られた，正規化が有利に働く状況下での性能を左右する要素についてより詳しく調べる．

4.3.1 定義域の各辺の密度と検索

各軸の幅を等しくした定義域内（すなわち，立方体）でのオブジェクトに対する検索に正規化がどのように働くかを示した表が**表3** と**表4** である．ここでは，NR*-tree が想定していない，軸間に差がない状況を再現し，正規化がどのように働くかを見る．

表4 では三次元の矩形オブジェクトが 20480×20480×20480 の立方体内に一様分布している（**図7** 参照）．三次元の矩形オブジェクトのサイズは '$l_1l_2l_3$' ($l_i(i=1,2,3)$ は $\{A,B,C\}$ のいずれか）の形で与えられ，その順に三次元の矩形オブジェクトの X，Y，Z 軸方向の辺の長さを表す．また今回の実験では，検索領域の大きさはオブジェクトの大きさを元に作られる．その倍率のパラメータは '$m_1m_2m_3$' ($m_i(i=1,2,3)$ は $\{a,b,c\}$ のいずれか）の値であり，領域の大きさは，したがって '$(m_1\times l_1)\times(m_2\times l_2)\times(m_3\times l_3)$' となる．また，表の中の数値は評価値であり，'平均' は行または列の評価値の平均である．たとえば，表4 における 'ABC' 行-'acc' 列の値は '0.77' である．これは '341×455×682' の直方体オブジェクト集合について，

表 3 定義域固定の環境下での性能比較 (1)
Table 3 Performance of indices on fixed domain (1).

オブジェクトの MBR のサイズ	検索領域のタイプ						
	aaa	aab	aac	abb	abc	acc	平均
AAA	1.07	1.06	1.15	1.07	1.17	1.18	1.11
AAB	1.24	1.21	1.08	1.19	1.06	1.06	1.14
AAC	0.97	0.93	0.85	0.91	0.8	0.86	0.88
ABB	0.92	1.01	0.99	0.92	0.9	0.89	0.93
ABC	1.15	1.04	1.05	0.98	1	1.07	1.04
ACC	0.94	0.94	0.9	0.92	0.91	0.88	0.91
平均	1.04	1.03	1	0.99	0.97	0.99	1

定義域: 10240 × 10240 × 10240
A: 170 B: 227 C: 341
a: ×2 b: ×5 c: ×8

表 4 定義域固定の環境下での性能比較 (2)
Table 4 Performance of indices on fixed domain (2).

オブジェクトの MBR のサイズ	検索領域のタイプ						
	aaa	aab	aac	abb	abc	acc	平均
AAA	1.07	1.14	1.07	1.14	1.06	1.13	1.1
AAB	0.95	0.96	0.9	0.93	0.87	0.85	0.91
AAC	1.06	1.02	1.02	1.04	1.03	1.03	1.03
ABB	0.98	1.13	1.19	1.06	1.12	1.22	1.11
ABC	0.74	0.77	0.82	0.81	0.83	0.77	0.79
ACC	1.05	1.04	1.03	0.98	0.96	0.93	0.99
平均	0.97	1.01	1	0.99	0.97	0.98	0.98

定義域: 20480 × 20480 × 20480
A: 341 B: 455 C: 682
a: ×2 b: ×5 c: ×8

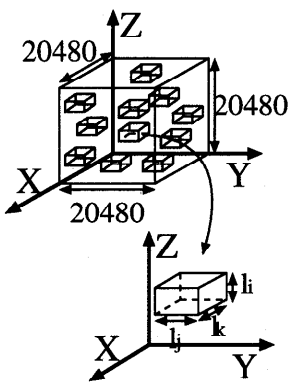


図 7 定義域とオブジェクトの例
Fig. 7 Sample of domain and objects.

'681×3640×5456' のランダムな位置にある直方体領域で行った 110 回分の領域検索から得られた評価値の平均として，評価値 '0.77' を得たことを意味する．

2 つの表は定義域を固定したうえで，オブジェクトの MBR の辺の長さを 3 種類，また検索領域も 3 種類を設定している．表 3 と表 4 の違いはオブジェクトの大きさである．

2 つの表の数値を見ると，ともに列内よりも行内の数値のばらつきが小さい．列内は検索に与える領域を，また行内はオブジェクトの領域の持つ形を表している．これから，NR*-tree の検索性能に与える影響はオブジェクトの領域と定義域の形との間に見られるといえる．さらに，この実験では限られた空間の中に決められた数のオブジェクトを作っていることから，オブジェクトの領域よりも各軸での込み具合を表しているとも考えられる．つまり，この NR*-tree を用いる利点がない状況下においては，正規化は良くも悪くもほとんど影響を与えていない．ただ，細長い軸に関して長い質問領域を用いる場合，言い換えると，オブジェクトに関して高い密度を持つ軸を長くとる質問領域に関しては，NR*-tree が有利な傾向が見られた．

4.3.2 軸間に差がある状況

次に定義域の辺の長さに偏りがある場合の結果を，**表 5** と**表 6**，**表 7**，さらに**表 8** にあげる．これらの実験での条件は正規化を導入するきっかけを再現するために，定義域を偏らせて固定したうえで，直方体のオブジェクトを定義域の相似形にし，検索効率に与える影響を調べる．

各表の列は矩形領域と定義域の種類を同時に表現している．たとえば，表 6 での 'XYZ' は，'1280×20480×327680' の大きさの直方体である定義域に '25×409×6553' の MBR を持つオブジェクトを分布させていることを意味している．また，先の実験と同様に検索領域としてオブジェクトの各辺を整数倍にしてできる領域を使う．

結果として，先の実験とほぼ似た評価値の傾向が得られ，定義域の長い軸に対して，検索領域の辺を長くとる質問であるほど（目安としては，表の下方に向かうに従い）良い評価値を得ている．表 5 と表 6 の 2 つの表を比べると，行と列の対応する部分では表 5 の 'XZZ' 行の '0.02' の差を除いたすべてで表 5 側が良い評価値を得ている．同様の関係は，表 7 と表 8 の間にも成り立っている．それぞれの組で異なっているのは，定義域内のオブジェクトの交差の度合いである．ここから，密度が高い程に NR*-tree の正規化が有効に働く点が分かる．また，表 5 と表 7 の 2 つの表を比較すると，各軸間での辺の長さの差が広がるにつれて NR*-tree の正規化が有効に働くことも見ることができる．

4.4 実験結果のまとめと正規化の得失

実験結果から，正規化の有効性は検索領域とオブジェクトの形に影響されることが分かり，また以下の傾向が確認された．

- 平均的なオブジェクトと定義域において特別に長

表5 定義域の辺の長さに差がある環境下での性能比較 (1)
Table 5 Performance of indices on squashed domain (1).

定義域のサイズ	検索領域のタイプ						
	aaa	aab	aac	abb	abc	acc	平均
XXX	0.93	0.95	0.97	1.13	1.16	1.03	1.02
XXY	1.29	1.03	0.86	0.99	0.84	0.86	0.97
XXZ	0.94	0.74	0.63	0.79	0.67	0.72	0.74
XYY	0.89	0.78	0.7	0.67	0.61	0.58	0.7
XYZ	0.79	0.61	0.53	0.55	0.47	0.43	0.56
XZZ	0.67	0.55	0.51	0.46	0.41	0.38	0.49
平均	0.91	0.77	0.7	0.76	0.69	0.66	0.74

定義域: X: 1280 Y: 20480 Z: 327680
obj: X: 32 Y: 512 Z: 8192
a: ×2 b: ×5 c: ×8

表6 定義域の辺の長さに差がある環境下での性能比較 (2)
Table 6 Performance of indices on squashed domain (2).

定義域のサイズ	検索領域のタイプ						
	aaa	aab	aac	abb	abc	acc	平均
XXX	1.19	1.01	1.03	1	1.04	0.97	1.04
XXY	1.24	0.98	0.79	1.06	0.88	0.93	0.98
XXZ	1.12	0.83	0.79	0.88	0.83	0.86	0.88
XYY	1.67	1.33	1.14	1	0.86	0.8	1.13
XYZ	1.02	0.74	0.66	0.6	0.54	0.48	0.67
XZZ	0.77	0.53	0.47	0.39	0.35	0.33	0.47
平均	1.16	0.9	0.81	0.82	0.75	0.72	0.86

定義域: X: 1280 Y: 20480 Z: 327680
obj: X: 25 Y: 409 Z: 6553
a: ×2 b: ×5 c: ×8

表7 定義域の辺の長さに差がある環境下での性能比較 (3)
Table 7 Performance of indices on squashed domain (3).

定義域のサイズ	検索領域のタイプ						
	aaa	aab	aac	abb	abc	acc	平均
XXX	1.01	1.07	1.06	1.11	1.1	1.07	1.07
XXY	1.15	0.9	0.94	0.91	0.95	0.97	0.96
XXZ	1.19	0.94	0.77	0.98	0.8	0.83	0.91
XYY	1.2	1.03	0.99	0.87	0.82	0.77	0.94
XYZ	1.1	0.86	0.74	0.74	0.65	0.66	0.79
XZZ	0.81	0.68	0.59	0.56	0.48	0.42	0.59
平均	1.07	0.91	0.84	0.86	0.79	0.78	0.87

定義域: X: 1280 Y: 10240 Z: 81920
obj: X: 32 Y: 256 Z: 2048
a: ×2 b: ×5 c: ×8

表8 定義域の辺の長さに差がある環境下での性能比較 (4)
Table 8 Performance of indices on squashed domain (4).

定義域のサイズ	検索領域のタイプ						
	aaa	aab	aac	abb	abc	acc	平均
XXX	1.01	1.05	0.97	1.12	1.06	1	1.03
XXY	0.92	0.85	0.81	0.94	0.89	0.91	0.88
XXZ	0.97	0.77	0.71	0.83	0.77	0.83	0.81
XYY	1.06	0.97	0.87	0.84	0.75	0.68	0.86
XYZ	1.02	0.8	0.68	0.73	0.63	0.62	0.74
XZZ	0.92	0.61	0.54	0.48	0.42	0.36	0.55
平均	0.98	0.84	0.76	0.82	0.75	0.73	0.81

定義域: X: 1280 Y: 10240 Z: 81920
obj: X: 40 Y: 320 Z: 2560
a: ×2 b: ×5 c: ×8

い軸が存在する場合に正規化が有効に機能する

- 上記において，質問領域がオブジェクトと定義域の長い軸に広い幅をとるほどに検索の効率が増す

以上から，オブジェクトが持つ辺と定義域に関して，特定の軸が突出している状況下の場合に NR*-tree の利用が有効となることが分かった．このインデックスの特性は，3.2 節にある例に適応するといえる．また，現実の二次元空間においても同様であり，平面的な幅のとる数値と高さのとる数値の間に差がある状況等においても NR*-tree は有効に働くと思われる．さらに，今回のインデックスでは正規化により時間と空間を融合したが，温度等の他の尺度の値を対象とすることも可能である．

5. 関連研究

空間インデックスにも様々な実現法や応用がある．我々の用いたデータ構造やオブジェクトの表現法等に関連する他の空間インデックスへのアプローチと我々のそれを比較する．

5.1 軸属性によりデータ管理法を変える方法

時空間インデックスとして，'位置情報によるデータ構造（空間木）' と '時間オブジェクトを管理するデータ構造（時間木）' の 2 つのデータ構造を用意する方法が仲ら[17]により提案されている．この手法では，問合せの種類を調べ，空間木と時間木のどちらを用いるかを動的に決定する．

同様に属性によるデータ管理を分離する別の方法で，単一のデータ構造を使う方法として，Kolovson ら[2]の Segment R-Tree があげられる．Segment R-Tree は，時空間領域のインデックスであるが，時系列で変化するデータ，たとえば労働者の給与の変遷等をも扱える．この時系列データをも扱おうとする目的は我々と共通しているが，実現のアプローチに違いがある．Segment R-Tree では時空間オブジェクトの表現として，時間を線分に，また時間以外の属性を矩形にする方法をとっている．属性により扱いを変える点が我々の方法と異なっている．

さらに，時間と空間を 1 つのデータ構造により扱う他の方法として Shen ら[3]による Time Polygon index（TP-index）がある．彼らの TP-index では，始点となる時刻および時間の幅を三角形中の位置で表現する点で Segment R-Tree と異なっている．時空間を属性で分離しない我々の方法は，時間と空間の両属性を区別せずに利用できる点で有利である．また，領域

検索アルゴリズムも単純である．加えて，このアルゴリズムは空間属性のみの場合にもそのまま適用できる．

5.2 MBR を直接用いない方法

定義域を等幅に分割したます目に区切り，ます目の集合により領域を表したうえで，領域検索を行う方法が提案されている[7]．この方法では，ます目の集合に z-order[6] と呼ばれる符号化を行い，文字列にする．データ構造には B-tree 等の文字列の扱いに適した構造を用いることができる．また，領域の交差は文字列比較により判定する．フィルタのみでなく，次の段階であるリファインメント部分での処理についても考えたうえで，良好な性能を持たせている点は参考にできる．

5.3 他の空間インデックスの応用

時間や空間にとらわれず，より高次元の空間を対象としたインデックスとして，空間インデックスを応用した例がいくつか存在している．X-tree[13] は R*-tree をベースにしている点や，時間や空間の属性にとらわれない点で我々の方法と共通する．X-tree では，中間ノードの持つ子ノード数（fanout）が可変であるノード（Supernode）を用意している．これに加え，中間ノード間の領域の重複をなくしている．しかし，次元が数十から百程度の領域を対象としている点が我々のアプローチと異なる．

この構造に正規化を導入すれば，我々の場合と同様に性能向上が期待できると思われるが，これは今後の課題である．

SR-tree[10] も R*-tree を基にした構造であり，高次元の点を対象とする．高次元になると距離が実際の'遠近関係'を正しく反映しない問題が R*-tree には存在する．この問題を解決するために，SR-tree では MBR の代わりに包囲矩形と包囲球を併用しており，結果として効率を改善できることを示している．この場合，データ自身を変更することなく，データの解釈を変える点で，我々とは異なる「正規化」を行っていると考えられる．

6. むすび

本論文では，時空間領域のためのインデックスに空間インデックスを応用するときの問題点を指摘するとともに，問題解決のための正規化の提案と性能実験による効果の確認を行った．

時空間インデックスには，構成の似た空間インデックスで確立されている方法を採用した．時空間インデックスでは，空間インデックスよりも時間に関する分だけ次元が高くなり，オブジェクト表現も複雑である．そのため，Disk I/O を減らすために時空間インデックスの重要性が増している．

本論文で提案する時空間インデックスは，幅を扱うインデックスとして有効である R*-tree を基本としている．しかし，R*-tree において，異なる属性，単位を持つ数値を直接比較することは効率的なクラスタリングの妨げになる．我々は，属性の異なる数値を用いる場合に問題となる Area，Margin と Length に正規化を導入する問題解決の一方法を提案した．正規化はインデックスの構築時のみに作用し，検索を実行する場面には影響しない．

正規化の性能改善に与える影響については，一様分布させたオブジェクトに対する計算機実験により確かめた．正規化の導入と直接関係しない，空間の軸幅が等しい状況下での実験では正規化による性能の変化は1%にとどまり正規化が性能を劣化させないことを示した．

逆に正規化に有利に働くと予想される，オブジェクトと定義域の両方で軸間での幅の差を大きくした実験では，軸間での幅の差が広がるに従い，正規化は検索性能を大きく向上させた．さらに，定義域と平均的オブジェクトともに突出した軸を持つ状況下で，この突出した軸に関して検索領域の幅を広くとるにつれて良い検索性能を見せる傾向も確認できた．

このように NR*-tree は，時空間オブジェクトを高速に処理する能力を持っており，時空間という特徴的な環境での問題の解決に有効であると考えられる．

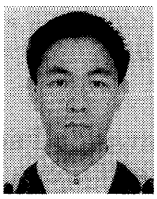
謝辞 適切かつ緻密なご指摘とご指導をいただいた査読者に感謝いたします．また，この研究の一部は文部省平成 10 年度科学研究費補助金重点領域研究（課題番号 08244105）の補助を受けている．

参考文献

1) Guttman, A.: R-trees: a dynamic index structure for spatial searching, *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pp.47–57 (1984).
2) Kolovson, C.P. and Stonebraker, M.: Segment Indexes: Dynamic Indexing Techniques for Multi-Dimensional Interval Data, *Proc. 1991 ACM SIGMOD*, pp.138–147 (1991).
3) Shen, H., Ooi, B.C. and Lu, H.J.: The TP-Index: A Dynamic and Efficient Indexing Mechanism for Temporal Databases, *10th International Conference on DATA ENGINEERING*, pp.274–281 (1994).
4) Horinokuchi, H., Kuroki, S. and Makinouchi, A.: Design and Implementation of R*-tree for Spatiotemporal Index, *Proc. IPSJ International*

Symposium on Information Systems and Technologies for Network Society, Fukuoka, Japan, pp.199–202 (1997).
5) Bentley, J.L.: Multidimensional Binary Search Trees Used for Associative Searding, *Comm. ACM*, Vol.18, No.9, pp.509–517 (1975).
6) Orenstein, J.: Spatial Query Processing in an Object-Oriented Database System, *Proc. 1986 ACM SIGMOD*, pp.326–336 (1986).
7) Orenstein, J.: A Comparison of Spatial Query Processing Techniques for Native and Parameter Spaces, *Proc. 1990 ACM SIGMOD*, pp.343–352 (1990).
8) Stonebraker, M., Frew, J., Gardels, K. and Meredith, J.: THE SEQUOIA 2000 STORAGE BENCHMARK, *Proc. 1993 ACM SIGMOD*, pp.2–11 (1993).
9) Beckmann, N., Kriegel, H.P., Schneider, R. and Seeger, B.: The R*-tree: An Efficient and Robust Access Method for Points and Rectangles, *Proc. 1990 ACM SIGMOD*, pp.322–331 (1990).
10) Katayama, N. and Satoh, S.: The SR-tree: An Index Structure for high-Dimensional Nearest Neighbor Queries, *ACM SIGMOD '97 5/96*, Tucson, Arizona, pp.369–380 (1997).
11) Cattell, R.G.G. (Ed.): *The Object Database Standard: ODMG-93*, Morgan Kaufmann (1994).
12) Cattell, R.G.G. (Ed.): *The Object Database Standard: ODMG-93 Release 1.2*, Morgan Kaufmann (1996).
13) Berchtold, S., Keim, D.A. and Kriegel, H.P.: The X-tree: An Index Structure for High-Dimensional Data, *Proc. 22nd VLDB Conference*, Mumbai (Bombay), India, pp.28–39 (1996).
14) Brinkhoff, T., Kriegel, H.P. and Schneider, R.: Comparison of Approximations of Complex Objects Used for Approximation-Based Query Processing in Spatial Database Systems, *9th International Conference on DATA ENGINEERING*, pp.40–49 (1993).
15) Sellis, T., Roussopoulos, N. and Faloutsos, C.: THE R^+-tree: A DYNAMIC INDEX FOR MULTI-DIMENSIONAL OBJECTS, *Proc. 13th VLDB Conference*, Bringhton, pp.507–518 (1987).
16) 黒木 進，牧之内顕文：位相空間データモデルUniverseでの空間，時間，時空間データ表現，情報処理学会論文誌（投稿中）．
17) 仲 篤起，才脇直樹，辻本浩章，西田正吾：高速検索のための時空間データ管理方式—状態が複数の場合の検討，信学技報，DE96-86 (1997-01), pp.73–78 (1997).

(平成10年 7 月21日受付)
(平成10年12月 7 日採録)



堀之口浩征（学生会員）
平成9年九州大学大学院システム情報科学研究科知能システム学専攻修士課程修了．現在同大学院博士課程2年．時空間データベースシステムに関する研究に従事．

黒木 進（正会員）
昭和63年東京大学工学部計数工学科卒業．平成2年同大学院工学系研究科計数工学専攻修士課程修了．同年九州大学工学部情報工学科助手．平成8年より同大学院システム情報科学研究科助手．マルチメディアデータベース，時空間データベースの研究に従事．

牧之内顕文（正会員）
昭和42年京都大学工学部電子工学科卒業．昭和45年グルノーブル大理学部応用数学科 Docteur-Ingènieur 取得．(株) 富士通，(株) 富士通研究所，九州大学工学部教授を経て，平成8年同大学院システム情報科学研究科教授．ACM, IEEE Computer Society 各会員．