D-028

大きさの異なる図形データのための空間データ構造管理方式

A management method of spatial data structure for multi-size drawings

出木原 裕順†
Hiroyuki Dekihara

1．はじめに

GIS や CAD の分野では空間データを効率的に管理することが必要不可欠である．空間データを効率的に管理する手法の一つに，kd 木[1]や R 木[2]，BD 木[3]，MD 木[4]などの木構造を応用した空間データ構造がある．これらの空間データ構造では，分割した空間に存在する空間データや，距離の近接性を基準にグループ化した空間データを木構造の葉やノードに対応させて管理することで，位置情報を基にした効率的な空間データへのアクセスを実現している．しかしながら，これらの空間データ構造では，形状の小さな空間データの中に形状の大きな空間データが混在していた場合，検索性能が低下してしまう問題があった．そこで，本稿では，この問題を改善する改良法を新しく提案する．提案法では，検索性能を悪化させる原因となる大きな図形は，通常の葉とは異なる葉で管理することで，検索性の性能の低下を抑えている．また，本稿では，主要な空間データ構造の 1 つである R 木を基に提案法を適用し，従来法との比較実験により，提案法の有用性を示している．

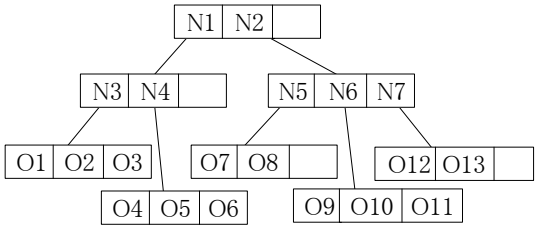
2．空間データ構造

2.1 R 木

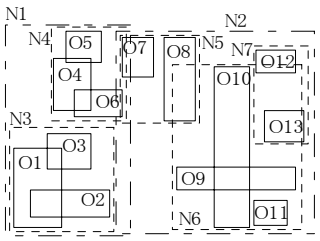
R 木は，B+木のデータ構造を応用した効率的な空間データ構造の 1 つである（図 1 参照）．空間データであるオブジェクトへのポインタを終端ノードである葉が保持している．葉の管理領域は，葉に格納されているポインタ先のオブジェクトを包含する最小の超長方形を形成している．この超長方形を最小外接矩形や MBR（Minimum Bounding Box）と呼ぶ．内部ノードは，子ノードへのポインタを保持しており，そのノードを頂点ノードとした部分木内のすべてのオブジェクトを包含する MBR を管理領域として保持している．葉が一杯になったとき，葉が保持しているポインタ先のオブジェクト群の分割，すなわち葉の分割が発生する．葉の分割では，分割した 2 つの葉の超長方形の領域が最小かつ領域同士の重なりが最小になるようにオブジェクトを 2 つのグループに分割する．範囲検索などを行う場合は，内部ノードの MBR を調べながら，R 木全体の頂点ノードである根から葉へと木構造をたどっていき，葉のポインタ先のデータへアクセスする．なお，R 木の詳細なアルゴリズムについては，文献[2]に詳しい．

2.2 大きな図形による性能の低下

R 木が管理している空間データの中に，その他のオブジェクトと比べて相対的に大きな図形が混在していた場合，空間データ構造の検索性能が悪化してしまう問題がある．例えば，図 1 の(b)の状態へ，図 2(a)のように新しいオブジェクト O14 が挿入された場合，O14 の大きな形状により，N2 や N7 の MBR が大きくなってしまう（図 2(b)参照）．



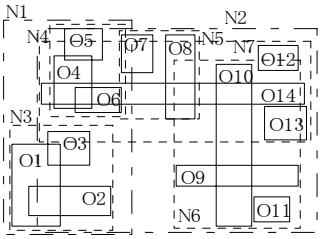
(a) 木構造



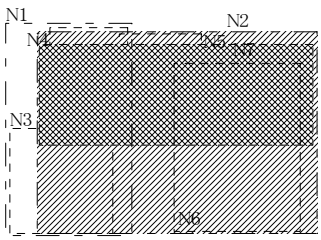
(b) 管理領域

図 1　R 木の例

その結果，検索時にノードの MBR を調べたとき，データが存在していない範囲を検索する場合でも，N2 と N7 が検索対象となってしまう可能性がある．これにより，データ構造の検索性能が低下してしまう．本稿では，この問題の改善を図っている．



(a) O14の追加



(b) N2とN7の管理領域

図 2　大きなオブジェクトによる管理領域の増大

† 広島国際大学工学部情報通信学科

3. 提案法

3.1 独立葉による大きな図形の管理

検索性能を悪化させる原因となりうる形状の大きな空間データを管理するために，終端ノードである通常の葉とは別に，独立した葉（独立葉）を準備する．独立葉の例を図 3 に示す．独立葉は，リスト構造で連結させ，空間データ構造とは別に構築する．独立葉では，単一軸に基づいてオブジェクトを並び替えて保持しておき，空間データ構造から独立葉へは，ハッシュテーブルを利用してアクセスする．独立葉が一杯の場合には新しい独立葉を作成し，リスト構造で連結する．入力された空間データの大きさが内部ノードや葉が管理する MBR の大きさと比べて基準値以上の場合には，独立葉に格納する．なお，実験では，独立葉への格納する基準は，オブジェクトの面積の大きさとし，葉や内部ノードの MBR の面積を基準値として調べ，基準値以上ならば独立葉に格納させている．

3.2 提案法のデータ構造

提案法の内部ノード **N** と葉 **L**，独立葉 **IL** のデータ構造，およびハッシュテーブル **H** をそれぞれ以下に示す．まず，内部ノード **N** のデータ構造を以下に示す．

N = *{E[M], MBR, p}*,

なお，*E[M]*は，エントリ **E**（子ノードへのポインタ，子ノードの MBR）の配列で *M* は最大子ノード数である．*MBR* は **N** の最小外接矩形，*p* はハッシュテーブルへのポインタである．次に，葉 L のデータ構造を以下に示す．

L = *{d[N], MVR, p}*,

なお，*d[N]*は，オブジェクトへのポインタ配列で *N* は，葉が格納できる最大オブジェクト数である．*MBR* は **L** の最小外接矩形，*p* はハッシュテーブルへのポインタである．独立葉 **IL** のデータ構造を以下に示す．

IL = *{d[N], p}*,

なお，*d[N]*は，オブジェクトへのポインタ配列で *N* は，葉が格納できる最大オブジェクト数である．*p* はリスト構造上の次の独立葉を指し示すポインタである．最後に，ハッシュテーブル **H** を以下に示す．

H = *{h[S]}*,

なお，*h[S]*は，エントリ **h**（独立葉へのポインタ，独立葉内の検索開始配列番号）の配列で *S* は最大格納数である．

4. 数値実験

4.1 実験条件

オリジナルの R 木（R）と提案法を実装した R 木（ER）を実装し，シミュレーション実験を行った。実験では，10

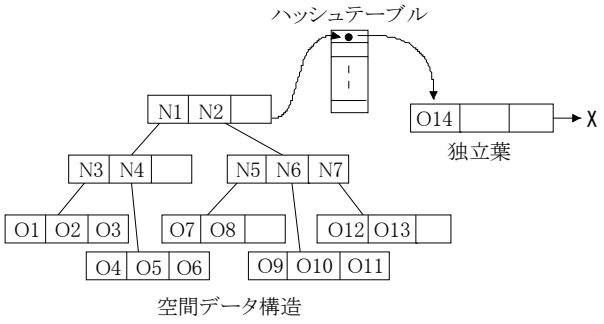


図 3　提案法の概要

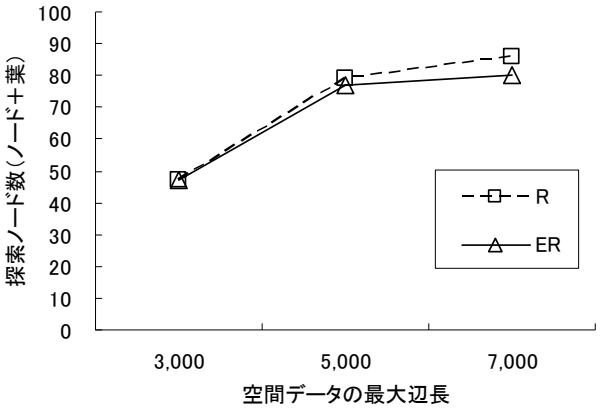


図 4　数値実験の結果

万個の空間データを 10,000×10,000 のデータ空間において管理する．また，空間データの 1 辺の大きさをデータ空間の 1 辺の 0.1-90.0%（すなわち，10-9,000）の範囲で指数乱数に従って変動させている。空間検索時にたどる内部ノードと調査する葉（や独立葉）の数で検索性能を比較検討する．

4.2 実験結果

R と ER の構築後に空間検索を実施した数値実験の結果を図 4 に示す．提案法は，オリジナルの R 木と比較して，検索時に探索するノード数（検索時にたどる内部ノードの数と探査する葉の数（独立葉を含む）の総和）は，最大で約 15%削減されている．また，最悪時でも，従来法と同程度である．以上より，提案法は R 木と比べて検索性能が効率的であることが確認できた．

5. おわりに

本稿では，小さな図形の中に相対的に大きな図形が混在しているような空間データ群を効率的に管理するための新しい空間データ構造の手法を新しく提案した．提案法では，検索性能を悪化させる原因となる大きな図形を管理するための独立した葉を用意して管理することで，空間データ構造の検索性能の悪化を抑えている．

参考文献

[1] J. L. Bentley, “K-d trees for Semidynamic Point Sets”, Proc. ACM Symp. on 6th SCG, 1990, pp. 187-197.

[2] A. Guttman, “A Dynamic Index Structure for Spatial Searching”, Proc. ACM SIGMOD, 1984, pp.47-57.

[3] 大沢裕，坂内正夫「2 種類の補助情報により検索と管理性能の向上を図った多次元データ構造の提案」，電子情報通信学会論文誌，Vol. J74, No. 8，1991, 467-475.

[4] Y. Nakamura, S. Abe, Y. Ohsawa and M. Sakauchi, “The MD-tree: An Efficient Data Management Strucuture for Spatial Objects”, IEEE trans. KDE., Vol.5, No.4, 1993, pp.682-694.