GBD 木のための空間インデックス高速初期構築法の提案

A Proposal of Fast Initial Spatial Index Construction Method for GBD-Tree

根岸 幸生♦ 大沢 裕♠

Yukio NEGISHI Yutaka OHSAWA

空間データ管理構造 GBD 木における空間インデックスの高速初期構築法を提案する．GBD 木は地理情報などの空間データを管理する上で効率の良いデータ管理構造であるが，その初期構築に長い時間を要していた．本稿ではバッチ処理的な手法による空間インデックスの構築法を提案する．これにより GBD 木の空間インデックス作成時間を 4 分の 1 に短縮した．また，本手法にて生成された空間インデックスは空間検索時にアクセスするオブジェクト数が従来手法より 40%ほど少なくなることを実験により示した．

The authors propose a new initial construction method for spatial index of GBD-Tree. GBD-Tree is an efficient data structure for management geographical entities. However, GBD-Tree needs long computation time for initial construction. 400% improvement in the speed is achieved by using proposal method. In addition, spatial indexes which created by the proposed method are 40% more efficient in spatial retrieval than previous method.

1. はじめに

現在，地理情報システムの分野では大縮尺数値地図データの整備がなされつつある．大縮尺数値地図とは2500分の1や500分の1といった程度の縮尺の大容量の数値地図であり，家形，道路沿，縁石，側溝といった詳細な地物まで記述されており，数値地図の容量も大きいことが特徴である．一方，筆者らの研究グループでは，時空間地理情報システム(STIMS)[3]を用いて地方自治体の日常的ないくつかの業務で使えるようなシステムのプロトタイプを提案している．これらのシステム上で大縮尺の数値地図を用いたいという要望も大きい．しかしながら，現在のSTIMS上で大容量の数値地図を用いたとき，STIMSの空間データ管理構造であるGBD木[1]のインデックスの初期構築にかなりの時間がかかるという問題がある．

筆者らは，これらの問題点を解決するために，バッチ処理的な手法による空間インデックスの初期構築法を提案する．また，空間インデックスの作成に要した時間と，この手法によって作成された空間インデックスの性能を実験により評価した．

2. GBD 木

GBD木[1]はR木[2]やk-d木と同様にデータの存在する空間を階層的に分割し，その分割過程を木構造で管理する多次元データ管理構造である．図1はGBD木の構造を示している．GBD木の木構造はM個のスロットを持つノードで構成されている．各スロットには子ノード(葉ノードでは図形データ自身)へのポインタと，その子ノードが対応する空間を表す領域式，および子ノードを根とする部分木中の全データを含む外接長方形(MBR)を持つ．領域式とは図形の代表点の空間上での位置を表現し，また分割された領域の位置と大きさを表現するものである．

R木とGBD木の違いは，前者ではデータ投入・削除と検索をMBRのみで行うのに対して，後者ではデータ投入・削除には図形のMBRの中心点に対応する領域式を用い，検索はMBRを参照しつつ行う点にある．したがって，GBD木はR木に比べてデータ投入・削除を高速化できる可能性がある．

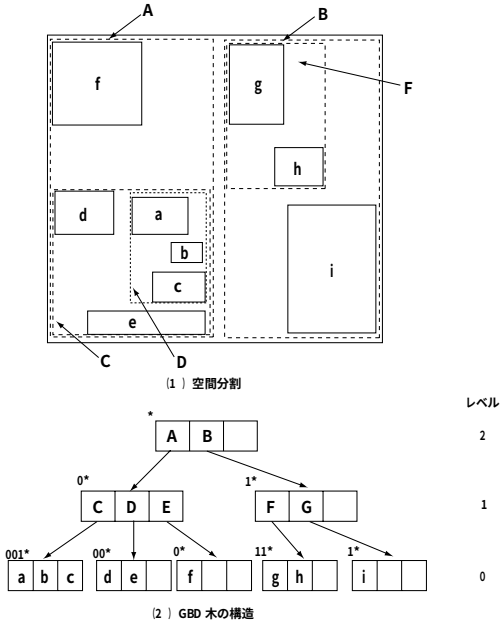

図 1 GBD 木の構造

Fig.1 Structure of GBD-Tree

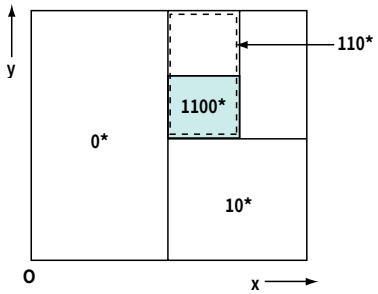

図 2 領域の分割例

Fig.2 An Example of Region Division

GBD木で管理する領域の形状は分割座標軸を巡回的に変えながら，面積を 2 等分にするように分割を繰り返すことにより得られる正方形，または縦横比が2:1の長方形である．この領域の位置と大きさを表すものが領域式である．領域式は「0」「1」およびパターンの終了を表す記号「*」で表現さ

♦ 学生会員 埼玉大学大学院理工学研究科情報数理科学専攻 negishi@mm.ics.saitama-u.ac.jp

♠ 正会員 埼玉大学工学部情報システム工学科 ohsawa@mm.ics.saitama-u.ac.jp

れる．図2はx軸，y軸交互に２等分割することにより作られる領域の例を示しており，各領域に対応する領域式を示している．空間分割過程と領域式の対応を以下に示す．

(1) 領域式中のビット「0」は空間を2等分割したときの座標原点に近い側の領域に，「1」は遠い側の領域に対応している．図2では，y軸と平行な軸で2等分割されたとき，領域式中のビット「0」は左半分，「1」は右半分になり，x軸と平行な軸で2等分されたとき「0」は下半分，「1」は上半分に対応している．

(2) 領域式の最も左側のビットが最初の空間分割に対応しており，右側に行くに従い小さな領域の分割に対応している．

また，領域式の大小関係は次のように定義される．

領域式の長さがn1，n2 (n1 > n2)の領域式R1，R2において

(1) R1の上位n2ビットがR2に一致するとき， R1 < R2とする．

(2) 上位n2ビット目において始めてR1とR2のパターンに不一致があるとき，一致しないビットの値が「0」の側が小さいものとする．

たとえばR1を「0110011*」，R2を「0110*」とすると，(1)より R1<R2である，またR1が「00100*」，R2が「00101*」の場合，両者の5ビット目が異なり，かつR1の5ビット目が「0」であることから(2)より R1 < R2である．

更に，領域式の包含関係を次の様に定義する．２つの領域式R1，R2においてn1＜n2であり，かつR1，R2の上位n1ビットのパターンが一致するとき，R1はR2を包含するという．

3. バッチ処理によるインデックス作成方式

従来，GBD木にデータを追加する方法として，図形データを一つずつ逐次投入してインデックスを作成する手法が提案されている[1]．しかし，この手法では大容量データの空間インデックスを作成する際に長い処理時間を必要とする．このためすべてのデータをあらかじめ領域式の大小関係でソートしておき，バッチ処理的に空間インデックスを生成する方法を提案する．

基本的な処理の流れを以下に示す．

(1)初期構築に先立ち，すべての図形オブジェクトのMBRの中心点の領域式を求め，その大小関係で昇べきの順にソートする．個々の図形オブジェクトが本稿で扱う単位である．

(2)3.1で述べる方式により，図形オブジェクトをノードのスロット最大数(M)以下の葉ノードに分割する．

(3)葉ノードの数がMを超える場合，3.2で述べる中間ノードの分割を行い，必要に応じて木のレベル数を順次増加させていく．

(4)最も上位レベルのノード(根ノード)でのスロット数がM以下となった時点で処理を終了する．

3.1 葉ノードへの分割

この処理は全ての図形オブジェクトがその MBR の中心点の領域式でソートされている１次元配列全体から開始して再帰的な分割を繰り返し，各々の個数が最大スロット数(M)以下のグループ（葉ノード）に分割していく処理である．この処理の流れを図３に示す．但し，この図で右側が葉ノード中のデータの領域式を示し，右側が分割過程を示している．

(1) 配列 L のスロットにあるすべての図形の領域式を参照し，それらの共通領域式を求める(図 3(a))．これは配列 L が領域式でソートされているとき，その最初と最後の要素の領域式の排他的論理和を求め，その結果の先頭ビットから０が続くビット数の領域式により求められる．例えば図３の例では領域式の共通部分は０ビットであり，共通領域式は＊（即ち０ビット領域式）である．

(2) 求めた配列Ｌの共通領域式の末尾に０を加えたものを分割のための領域式(DD)とし，この領域式で配列Ｌを仮に分割する．ノードＬのスロット一つ一つに対し，分割のための領域式(DD)に各スロットの図形の領域式が含まれているか確認し，含まれているものとそうでないものとの２つのグループに分ける．例えば図3(b)の例ではDDが0*であり，それに含まれるものが１から９，含まれないものが１０から１２である．

Case Ⅰ．２つのグループに分割された両方のグループのスロット数が共にＭを超え，かつ両方のグループのスロット数の比率が 1:2 から 2:1 の範囲にあるとき，各々のグループに対して(1)以下の処理を再帰的に繰り返す．

Case Ⅱ．それ以外の場合，即ち ２つのグループに分けたうちの数の少ない側が，最大スロット数を下回っていた場合か両グループのスロット数の比率に大きなアンバランスがある場合，よりバランスの良い分割を目指して以下の処理を行う．

まず，この場合の例を示す．図 3(b)では 0*で分割を行うことにより，９個のグループと３個のグループに分けられる．そこで９個のグループに対して更に(1)以下の処理を実行すると，(c)に示すように 00*の領域式で６個のグループと３個のグループに分けられる．ここで 00*の領域式により，葉ノード全体を 00*に包含されるグループとそれ以外のグループに分割し，それぞれのグループに対して(1)以下の処理を再帰的に適用する．

以上の処理を行う結果，各葉ノードのスロット占有率の下限は文献[1]の条件が成立する．即ち，全ての葉ノードのスロット使用効率は，(M+1)/3 以上が保証される．

図３　葉ノード分割の例

Fig.3 Leaf Node Division Process

3.2 中間ノードの分割

葉ノードの数が最大スロット数(M)を超える場合，上位の非葉ノードにおいてもノードの分割が必要となる．この方式は文献[1]で述べられているものと同様であり，次の手順で実行される．

(1) 分割の対象となる非葉ノード (T)の各スロットの領域式を参照しつつ．その領域式が自分を含めいくつのスロットを内包しているか調べる．

(2) 最も(M+1)/2 に近い数を内包する領域式を新しい領域 A

の領域式とし，領域 A に内包される子ノードと，それ以外の子ノードに分割する．

この処理が順次上位のノードに対して実行されていき，最終的にある上位ノードにおいて，スロット数が M 以下となった時点で分割処理を終了する．そのノードが根ノードになる．

また，GBD 木では各ノードが，そのノードを根ノードとする部分木中に含まれる全ての図形を内包する MBR を保持している．ノードに対する MBR の付与は，分割処理が終了後，一括して実行される．

4. 性能の評価

4.1 各手法による性能評価

前章にて述べた手法を実装し実験により性能を評価した．実験では当研究グループで開発している，時空間情報管理システム:STIMS[3]の空間データファイルを読み込み，空間インデックスファイルの作成にかかった時間を測定した．実験には，800KB から 120MB までの４種類の数値地図を用いた．数値地図名，データサイズ，オブジェクト数を表 1 に示す．計算機への実装では領域式を 64 ビットとし，領域式の長さを 8 ビットで表現した．実験には，今回提案した手法と従来の STIMS で採用されている手法[1]の２種類について，それぞれ空間インデックス作成に要した時間を測定した．測定に用いた実験環境を表 2 に示す．

実験結果を表 3 に示す．また，本実験では，GBD 木の最大スロット数(M)を 50 とした．このスロット数は，現在 STIMS で用いられている最大スロット数である．

提案手法は，どの数値地図においても良好な結果が得られていることがわかる．また，従来手法との比較では，どの数値地図においても 4 倍程度速度が向上した．

表 1　実験に用いた数値地図
Table 1 Used Digital Maps in Experiment

数値地図	縮尺	データサイズ	オブジェクト数
JMCマップ KS5339	1/200,000	840KB	8,583
数値地図 25,000 さいたま市	1/25,000	1.4MB	27,074
トロピカルテクノセンター 那覇市	1/2,500	10MB	151,699
J-Mapple 200,000 日本全国	1/200,000	120MB	610,756

表 2　実験に用いた計算機のスペック
Table 2 PC Spec.

CPU	Pentium 4 2.8CG
RAM	2GB
OS	Windows XP Professional

表 3　空間インデックス作成に要した時間
Table 3 Result of Construction Time

ベースマップ	従来手法	提案手法
JMC KS5339	2.3	0.6
JSGI さいたま市	7.0	1.8
TTC 那覇市	39.4	10.4
J-Mapple 200000	164.4	44.3

単位[秒]

4.2 最大スロット数を変化させた場合の性能評価

次に，ノードあたりの最大スロット数を変化させた場合についての評価実験を行った．実験に用いた数値地図は先の実験で用いたトロピカルテクノセンターの那覇市のデータを用いた．測定は今回提案した手法と，従来の手法[1]の２つについて，最大スロット数を 25 から 1000 まで変化させインデックスの構築に要した時間を測定した．この実験結果を図 5 に示す．

提案手法は，スロット数が多くなるに従い処理時間が緩やかに減少している．一方，従来の手法ではスロット数が多くなるに従い処理時間が増加することがわかる．この結果は，提案手法ではすでにすべてのスロットを領域式でソートし，その後ノード分割をする．このため，スロット数が増えるに従いノード分割数が減り，処理時間が減少するためである．

一方，従来の手法ではデータを投入するごとに，木をたどり，適切な位置のノードを下る必要があるため，スロット数の大きいノードではノードにおける下位ノード選択処理に時間がかかり，構築時間が増加する．

実験結果より，提案手法は従来手法より高速に GBD 木のインデックスを作成することができる．特に GBD 木のノードの最大スロット数が多い場合には，提案手法がより高速にインデックスを作成することができる．

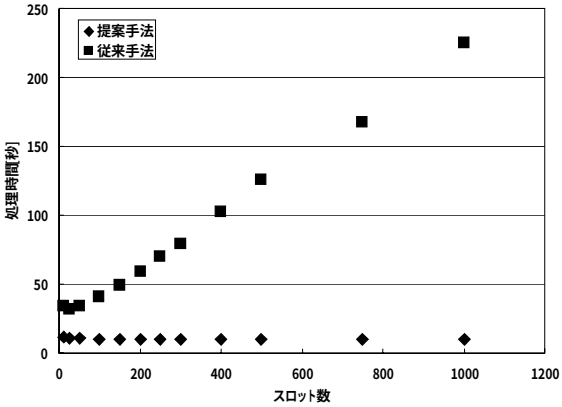

図 4　最大スロット数を変化させた場合における空間インデックス生成時間

Fig.4 Spatial Index Construction Time According to Maximum Slot Number Change

4.3 検索性能の評価

逐次的なインデックス作成方式と，提案手法の両者のインデックスの性能を比較するため 3 つの実験を行った．まず，それぞれの方式によって作成されたインデックスのスロットの占有率を比較した．加えて，空間データの検索時にたどったノード数と，アクセスしたオブジェクト数を比較する実験を行った．さらに，本手法に適したスロット数を求めるため，スロット数を変更した場合のインデックスサイズの変化についても調べた．

4.3.1 スロット占有率の比較

従来手法との性能を比較するため，スロットの占有率を比較した．スロット占有率とは，ノードの全スロットのうち子ノードや空間データに割り当てられている有効なスロットの占める割合である．この実験では，逐次投入方式とバッチ処理のそれぞれの方式で作成した空間インデックスをたど

り，各ノードにおいて有効なスロット数を調べ，スロット占有率を求めた．空間データにはトロピカルテクノセンターの那覇市のデータを用いた．また，スロット数(M)は 50 とした．

従来手法ではスロット占有率の平均値が 60.2%であったのに対し，提案手法ではスロットの占有率が 67.2%に向上した．この理由は，逐次投入方式ではノード分割をした後にも空間データが挿入されることで，最適な空間の分割ができず，分割後のノードに偏りができてしまうのに対し，バッチ処理ではあらかじめすべての空間データが分かっていることにより，バランスよく空間を分割できるためであると考えられる．

4.3.2 検索効率の比較

提案手法によって作成したインデックスの検索効率を調べるため，評価実験を行った．実験内容は，数値地図中にあらかじめ無作為に 100 箇所の地点を設定し，その範囲を中心とした 10,000×10,000 の正方形領域の検索処理をしたときの，たどったノード数とアクセスしたオブジェクトの数を調べ，その平均値を求めた．この正方形領域は数値地図の全空間サイズの約 0.1%にあたる，数値地図には先の実験と同じ那覇市のデータを用いた．数値地図の幅は 10,200,000，高さは 8,000,000 である．また，スロット数は 50 で実験をした．この実験結果を表 4 に示す．

表 4　空間インデックスへのアクセス数
Table 4 Accessed Nodes and Objects Number

	従来手法	提案手法
オブジェクトアクセス数	5654	3804
葉ノードアクセス数	184	109
中間ノードアクセス数	108	78

従来手法の逐次投入に比べ，提案手法ではたどるノード数，アクセスするオブジェクト数ともに平均値で 40%ほど少ない．

4.3.3 スロット数とインデックスサイズの比較

提案手法で作成されるインデックスに適したスロット数を調べるため，スロット数を変えながら GBD 木のインデックスを作成し，そのインデックスサイズを比較した．実験内容は，今回提案した手法を用いて，最大スロット数を 25 から 2000 まで変えてインデックスを作成し，GBD 木のインデックスのサイズを測定した．数値地図には先の 2 つの実験と同じ那覇市のデータを用いた．実験結果を図 6 に示す．

実験結果より，最大スロット数が増加すると，インデックスサイズは増加減少を繰り返しながら，緩やかに減少する結果が得られた．この現象は次のように説明できる．今回提案したバッチ処理的な分割手法では，最も理想的なデータが与えられた場合は，ノードは常に二等分に分割されていく．このような理想的な状況では，最大スロット数(M)を

$M=(\text{全空間オブジェクト数}/\ 2^k)\ (k = 1,2,3\cdots)$

にすると，スロット占有率は 100%になる．実際の分割では上記のような理想的な分割は起こりえないが，分割は平均的には 2 等分割に近い分割がされていると考えられるため，最大スロット数を上記の値にすることで，高いスロット占有率が得られ，インデックスサイズが減少する結果が得られる．

5. まとめ

本稿では，GBD 木の空間インデックスの初期構築を高速化するための手法について述べた．GBD 木の各ノードは，データ投入や削除のための補助情報である領域式と，データ検索のための子ノードをすべて包含する MBR を持つ．この 2 つの補助情報により，検索性能を犠牲にすることなく，データ更新を高速に行うことができる．本稿では，GBD 木の空間インデックスを高速化するため，すべてのオブジェクトを領域式順にソートし，領域式の共通部分を求め，再帰的に分割を繰り返すことによるバッチ処理的な手法で GBD 木の空間インデックスを作成する手法を提案した．また，本手法を実際の計算機上にインプリメントし，時空間情報管理システム(STIMS)の空間インデックス作成に要する時間を測定した．

提案手法は従来の手法より 4 倍程度高速にインデックスを構築することが可能である．またスロット数が増えた場合にはさらに高速に構築できる．このため，大規模データや精度の高い大縮尺の地図データの空間インデックスを作成する場合には本手法が有用である．

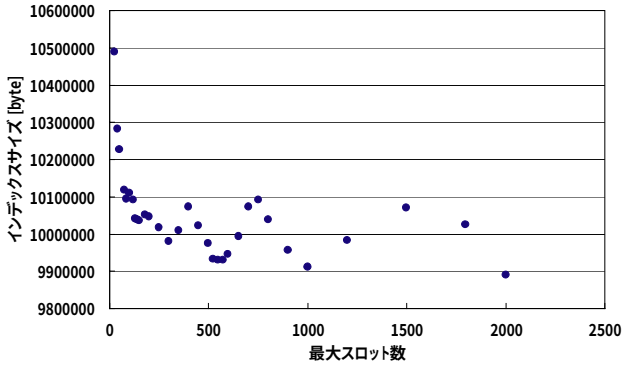

図 5　最大スロット数と空間インデックスサイズの関係

Fig.5 Index size of GBD-Tree

[文献]

[1] 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，J74-D-1，No.8，pp.467-475 (1991).

[2] Antonin Guttman: “A Dynamic Index Structure for Spatial Searching”, Proc. of the 1984 ACM-SIGMOD International Conf., pp.47-57 (1984).

[3] 根岸幸生，青木秀晃，笠原直，郭薇，川崎洋，大沢裕: 時空間管理のための地理情報システム STIMS，情報処理学会研究報告，2003-DBS-131(I)，pp.195-202 (2003).

[4] 根岸幸生，大沢裕：GBD 木のための空間インデックス高速初期構築法の提案，情報処理学会研究報告，2005-DBS-137(I)，pp.55-60 (2005).

根岸 幸生　Yukio NEGISHI

埼玉大学大学院理工学研究科博士後期課程在学中．2004 埼玉大学大学院理工学研究科博士前期課程修了．地理情報システムの研究・開発に従事．情報処理学会学生会員．日本データベース学会学生会員．

大沢 裕　Yutaka OHSAWA

埼玉大学工学部情報システム工学科教授．1978 年信州大学大学院工学研究科修了．工学博士．地理情報システムの研究・開発に従事．地理情報システム学会理事．電子情報通信学会，情報処理学会，映像情報メディア学会各会員．ASGIS steering committee．著書に「画像データベース」(昭晃堂)など．