空間データにおける 2^n 分割木を用いた m-最近傍キーワード検索

邱　　原† 大森　匡† 新谷　隆彦†

† 電気通信大学大学院情報システム研究科　〒 182–8585 東京都調布市調布ケ丘１－５－１
E-mail: †{kyuu __ genn,omori}@hol.is.uec.ac.jp, ††shintani@is.uec.ac.jp

あらまし　本稿は空間データにおける mCK(m-Closest Keywords:m-最近傍キーワード) 検索問題 [4] を扱う．先行研究である Zhang らの bR*-tree データ構造を用いた方式 [4] はノードの組合せと枝刈りを交互に行って, 探索空間を減らすが, いくつか制約がある．すなわち,(1) 頻繁に更新するデータに対して能力はよくない.(2) データセットのオーバラップには耐えられない. (3) 検索対象になるキーワードの集合が事前に決まっている必要がある．この問題に対して, 本稿は空間を $2^n \times 2^n$ の領域に分割する木構造 (2^n 分割木) を用いた探索アルゴリズムを述べ, 上記 (1)〜(3) を解決し, かつ, 枝刈り戦略を工夫して, 効率の向上を目指す.

キーワード　空間データ検索 m-最近傍検索

2^n split tree for m-Closest Keyword search on spatial data

Yuan QIU†, Tadashi OHMORI†, and Takahiko SHINTANI†

† The University of Electro-Communications,Graduate School of Information Systems
E-mail: †{kyuu __ genn,omori}@hol.is.uec.ac.jp, ††shintani@is.uec.ac.jp

Abstract　This work addresses the problem of m-Closest Keywords search(mCK search). A previous study by Zhang et al[4] uses bR*-tree for this problem. This scheme efficently reduces the search space but has some restrictions: (1)It is not good for frequently updated data. (2)It is not applicable for the overlap operation of data sets. (3)It must know the keyword set before the bR*-tree's construction. Thus we use the 2^n split tree and propose a corresponding algorithm to overcome these restrictions and apply some pruning strategies to improve the efficiency.

Key words　Spatial data search m-Closest Keywords search

1. はじめに

1.1 研究の背景と目的

近年,Web 上での検索には空間的制約を含む検索の割合が高く [1]，空間検索はサーチエンジンにおいても重要になっている．空間データへの新しい検索能力も重要である．例えば, 空間属性とテキスト属性を両方含むデータ集合に対した空間キーワード検索が注目されている [2][3]．ユーザはいくつかのキーワードを入力して，空間とテキスト両方を関連する情報を探索する方式である．Zhang らが提出した mCK 検索 [4] はこの方式の一種であり,m 個のキーワードを指定して, この m 個のキーワードに関連する一番近いオブジェクトセットを探すという問合せである．この検索の応用例として, 花火大会で「花火の場所」,「食事処」,「ショッピングセンター」の三つのキーワードを考えれば, 花火も見えるし, 食事もショッピングもできる最も小さいエリアを検索できる.

一方で,GPS が普及するに伴って，様々な位置情報の付いているデータ (テキストや写真など) がデータセンターに集まって，新たなサービスが提供されている (microsoft asia の GeoLife[5] や GeoTag,Twitter[6])．このようなサービスへの適用も可能にしたい.

本稿では，mCK 検索をデータの頻繁な更新やデータセットのオーバラップには耐えられるように改良する．すなわち，グリッドデータ構造に基づいて空間を $2^n \times 2^n$ の領域に分割する木構造 (2^n 分割木) を用いてデータを整理して，各キーワードに関連するノードの入れ子ループで m 次元ノードセットを組み合わせるというストラテジーを考える．そして，二つの枝刈りルールを利用して探索空間を減少するアルゴリズムを提案する．また，キーワード近似 MBR を利用してアルゴリズムの最適化を試し，効率を向上する.

1.2 問題の定義

空間データ集合における mCK 検索とは m 個のキーワードに関連する「一番近い」にあるオブジェクトセットを探すことである．オブジェクトセットの「近さ」は定義 1 の直径によって判断される [4].

定義 1(直径)：m 個のオブジェクト $t_1, t_2, ..., t_m$ により構成されたオブジェクトセット T に対して，セットの直径は式 (1) によって示す

$$diam(T) = \max_{t_i, t_j \in T} dist(t_i, t_j) \quad (1)$$

式の $dist(t_i, t_j)$ はオブジェクト t_i と t_j の間の距離 (ここではのユークリッド距離にすることになっている) である．

つまり，オブジェクトセットの直径は，セットの中にある任意の二つのオブジェクトの間の距離の最大値である．この直径が小さければ小さいほど全オブジェクトが近いと言える．

定義 1 によると，mCK 検索は実際に m 個のキーワードに対して，それぞれのキーワードに関連する一つのオブジェクトを取って m 次元のセットを組み合わせる．そして，これらオブジェクトセットの内，直径が一番小さいものを選ぶ問題になる．

定義 2(mCK 検索)[4]：d 次元空間のオブジェクト集合を与えて，各オブジェクトは $t(l_1, l_2, ..., l_d, w)(l_1, l_2, ..., l_d$：各次元の座標,$w$：関連するキーワード) の形式とする．$m$CK 検索は m 個のキーワード $\{w_1, w_2, ..., w_m\}$ のクエリ Q に応じて，m 次元のオブジェクトセット $T = \{t_1, t_2, ..., t_m\}(t_{i.w} \in Q$, また $t_{i.w} \neq t_{j.w})$ に, 直径は最小なものを探すことである．

例えば，図 1 は地図にキーワード $\{A, B, C\}$ に関連するオブジェクトの分布を表す．$\{A, B, C\}$ をクエリとして mCK 検索を行うと，丸印で囲んだ三つのオブジェクトによるセットが結果となり，目的のエリアも決めることができる．

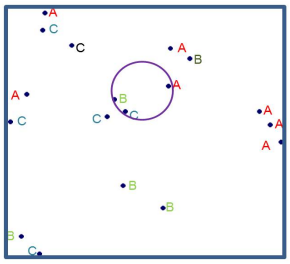


図 1　mCK 検索の例

2. 先行研究との違い

mCK 検索について，先行研究とした D.Papadias らの MultiWay Spatial Joins(MWSJ) 方式 [7] と Zhang らの bR*-tree データ構造を用いた方式 [4] がある．R*-tree に基づいたインデックスを使ってデータをフィルタリングすることである．MWSJ はキーワード毎に R*-tree を構築してから入れ子ループにより NN 探索を行なう方式である．bR*-tree は R*-tree のノードに各キーワードの含む情報と最小包囲矩形 (MBR) を記録して，検索の場合キーワードにかかわらず，根ノードから木をトラバースしながら，Apriori のストラテジーでノードセットの展開と枝刈りを交互に行って，探索空間を減らす．

しかし，MWSJ と bR*-tree 方式も R*-tree に基づいたデータを構築するので，位置情報のついたツイタのようにデータが頻繁に更新される場合，ノードのスプリットが多く，更新能力はよくない．また，クエリのキーワード数が多い場合 bR*-tree 方式の枝刈り能力は高いが，サーチエンジンの観点からいくつかの制限がある．まず，bR*-tree 方式は複数のデータセットのオーバラップを使った mCK 検索には耐えられない．例えば，ホテルデータセットと観光所データセットは別の bR*-tree で格納する場合，同時にこの二つのデータセットに mCK 検索を行うと，うまくできない．次に，検索の対象になるキーワードの集合が事前に決まっている必要がある．そのため，本稿は空間を $2^n \times 2^n$ の領域に分割する木構造 (2^n 分割木) を用いてデータの更新とオーバラップ問題に対処することを考え，枝刈りを工夫して探索効率を上げる．

3. 2^n 分割木

2^n 分割木は四分木のように空間を再帰的に $2^n \times 2^n$ の領域 (セル) に分割するのに使われる．もし n は 2 の場合，16(=4×4) 分割木になる．図 2(b) は同図 (a) のデータ分布に対応する 16 分割木である．各セルの情報は木のノードに格納される．各ノードに対して，ノード ID という番号を与える．ノード ID はヒルベルトや z-order など空間充填曲線による番号がつけてもよいが，ここでは簡単に単純な左→右と上→下の順番で番号をつける．また，ノードの中にデータがあるのにかかわらず，ノード番号はフル 16 分割木に対応している．そのため，再帰的にノード ID と 16 の割り算で得た商と余りを用いてセルの位置が確定される．

ノード ID に加えて，すべての中間ノードにキーワードごとキーワード分布ビットマップという情報もつけられている．キーワード分布ビットマップとは $2^n \times 2^n$ をサイズとして，それぞれのビットは一つの子ノードを代表する．

- 値が「1」の場合 該当キーワードが対応した子ノードに含まれる．
- 値が「0」の場合 該当キーワードが対応した子ノードに含まれない．

図 2(b) によると，ノード ID が「0」(根ノード) のノードにキーワード A のキーワード分布ビットマップは (1100 0000 0000 0000) であるので，根ノードの 1 番，2 番二つの子ノード，つまりノード ID が「1」,「2」のノードがキーワード A を含んでいて，他の子ノードが A を含まないと表示する．このキーワード分布ビットマップを利用して，任意のキーワード k に対して，根ノードから葉ノードまで k を含む子ノードへ辿ってから，全ての k に関連するノードを探すことができる．

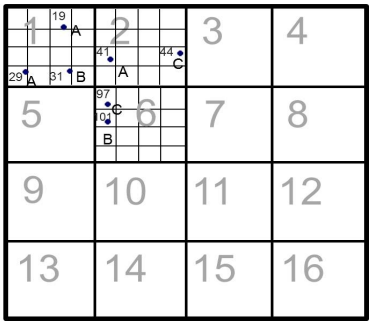
そこで，2^n 分割木のノードの形式は以下で表示する．

葉ノード　:(ノード ID, データページへのポインタ)
中間ノード　:(ノード ID, 子ノードへのポインタ, キーワード分布ビットマップ)

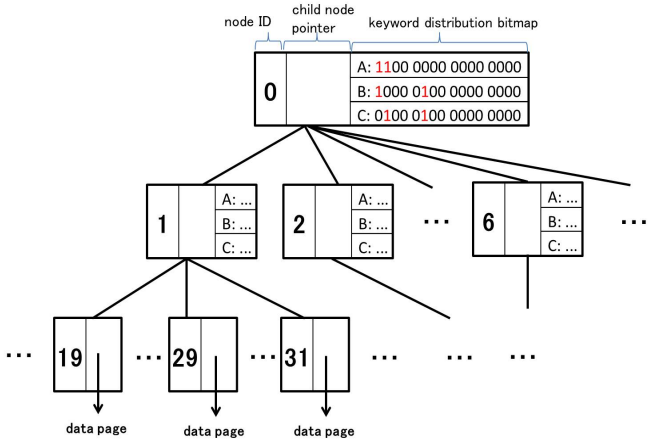
4. 検索アルゴリズム

4.1 準　　備

mCK 検索の主要な動作はそれぞれのキーワードに関連する



(a) データの地理分布



(b) 16 分割木のデータ構造

図 2　mCK 検索の例

データを組み合わせることである．そのため，2^n 分割木を使う mCK 検索はデータの組み合わせ数を減少する必要がある．本稿は 2^n 分割木をトラバースしながら，それぞれのキーワードに関連するノードを先に組み合わせて，ノードの間の距離で不要なデータの組み合わせを排除することを方針としてアルゴリズムを提案する．具体的には m 個のキーワードのクエリ $Q = \{w_1, w_2, ..., w_m\}$ に対して根ノードから，木の階層性質を利用して，各階層でそれぞれのキーワード $w_i (i = 1, 2, ..., m)$ に関連するノード $n_{i_j} (j = 1, 2, ...)$ を組み合わせ，m 次元のノードセット $\{n_{1_a}, n_{2_b}, ..., n_{m_c}\} (a, b, .., c \in j)$ を作る．そして，ノードセットを探索空間候補項にする．そこで，探索空間は「木の幅」と「木の深さ」二つの方向で生成する．「木の幅」のほうは同じ階層で m 個のキーワードを入れ子ループでノードを組み合わせることであるが，「木の深さ」のほうは木の階層を伸びることである．例えば，図 3 は図 2(b) の 16 分割木によって生成した探索空間木である．「木の広さ」と「木の深さ」で探索空間の生成は図の横軸と縦軸の方向に表す．

探索空間木からみると，16 分割木を用いて mCK 検索は状態空間探索の問題になる．本稿は探索空間木に深さ優先探索しながら，枝刈りを行うアルゴリズムを設ける．

枝刈りの判断基準を説明するためにいくつかの定義を与える．

定義 3(閾値 δ^*):δ^* は今まで計算した結果の中に一番小さい直径である．

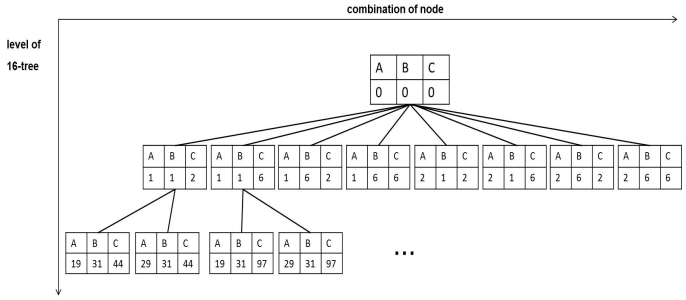


図 3　探索空間木

δ^* の初期値は対象となる全体空間の対角線の距離であり，つまり可能な直径の最大値である．δ^* は探索するとともに，減少する．

定義 4(二つのノードの $MINDIST$ と $MAXDIST$):図 4 によると，二つのノード n_i と n_j を与えて，$MINDIST(n_i, n_j)$ は n_i と n_j に代表された矩形区域の最小の距離であり，$MAXDIST(n_i, n_j)$ は n_i と n_j に代表された矩形区域の最大の距離である．

図 4　$MINDIST$ と $MAXDIST$

隣接する二つのノードの $MINDIST$ は 0 となっている．

定義 5(ノードセットの $MAXMINDIST$):
$MAXMINDIST(N)$ はノードセット N の中に任意の 2 つのノードの $MINDIST$ の最大値である．m 個のノード $\{n_1, n_2, ..., n_m\}$ に構成された m 次元のノードセット N に対して，$MAXMINDIST(N)$ は式 2 に示す．

$$MAXMINDIST(N) = \max_{n_i, n_j \in N, i \neq j} (MINDIST(n_i, n_j)) \quad (2)$$

$MAXMINDIST(N)$ はノードセット N に対応しているデータセットの直径の下限である．例えば，図 5 によって，キーワード A, B, C に関連するノードセット $N = \{n_A, n_B, n_C\}$ に対して，$MAXMINDIST(N) = MINDIST(n_A, n_B)$，つまり $dist(d_A, d_B) \geqq MAXMINDIST(N)(d_A$ は任意の A に関連するデータ，d_B は任意の B に関連するデータ) は必ず成立するので，ノードセット $N\{n_A, n_B, n_C\}$ により生成される答えの直径は $MAXMINDIST(N)$ 以上である．

4.2　枝刈りのルール

枝刈りの基準として，本稿は二つのルールを利用する．主に閾値 δ^* とノードの距離を比較して判断する．探索空間木の要素としたノードセットを判断して，もしルールを満足する場合，該当要素または要素を根ノードとしたサブ探索子空間を探索空

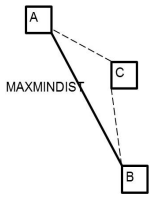


図 5　$MAXMINDIST$

間木から枝刈りすることができる．以下の二つの枝刈りルールを説明する．

[ルール 1]:クエリ $Q = \{w_1, w_2, ..., w_m\}$ に対応したノードセット $N = \{n_{w_1}, n_{w_2}, ..., n_{w_m}\}(n_{w_i}$ ：キーワード w_iに関連するノード$, i = 1, 2, .., m)$ について，もし $MINDIST(n_{w_i}$, $n_{w_j}) > \delta^*$ $(n_{w_i}, n_{w_j} \in N)$ が成立すれば，ノードセット N が排除される．つまり，ノードセットに，ある二つのノードの間の $MINDIST$ が δ^* より大きければ，該当ノードセットが排除される．

証明:n_{w_i} と n_{w_j} はキーワード w_i と w_j に関連するので，もし $MINDIST(n_{w_i}, n_{w_j}) > \delta^*$ であれば，すべてのキーワード w_i に関連するデータ d_{w_i} とキーワード w_j に関連するデータ d_{w_j} に対して，$dist(d_{w_i}, d_{w_j}) > \delta^*$．結果としたデータセット $\{..., d_{w_i}, .., d_{w_j}, ...\}$ の直径 $> \delta^*$ は必ず成立するので，ノードセット N が排除される．

[ルール 2]:クエリ $Q = \{w_1, w_2, ..., w_m\}$ に対応したノードセット $N = \{n_{w_1}, n_{w_2}, ..., n_{w_m}\}(n_{w_i}$ ：キーワード w_iに関連するノード$, i = 1, 2, .., m)$ について，もしノード n_{w_i} に対して，$MAXDIST(n_{w_i}, n_{w_t}) \quad < MAXMINDIST(N)(t = 1, 2, ..., m$ 且つ $t \neq i)$ が全部成立すれば，ノード n_{w_i} をノードセット N から一時的に削除する．

例えば，図 6 により，キーワード A, B, C に関連するノードセット $N = \{n_A, n_B, n_C\}$ に対して，
$MAXMINDIST(N) = MINDIST(n_A, n_B)$
$MAXDIST(n_C, n_A) < MAXMINDIST(N)$
$MAXDIST(n_C, n_B) < MAXMINDIST(N)$
直径は A と B に関連するデータの間の距離で決まると確定して，C に関連するデータとは関係ないので，子ノードの組み合わせまたはデータの組み合わせにおいて，C に関連するノードまたはデータを以降の δ^* の計算対象から外す．

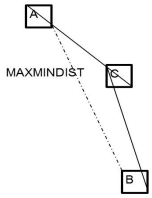


図 6　ル ー ル 2

4.3　アルゴリズム

2^n 分割木を用いた mCK 検索は木を深さ優先探索でトラバースしつつ，m 個のキーワードを入れ子ループでノードを組み合わせることであるので，アルゴリズムの設定について，再帰な方法で木のトラバースを行うが，非再帰のループでノードの組み合わせを実現する．具体的なアルゴリズムを Algorithm 1 に示す．このアルゴリズムでは，
1-5 行目:もしノードセットの要素は全部葉ノードの場合，各ノードに対応したキーワード関連するデータを取って，網羅的にデータセットを組み合わせて最小の直径を求める．もし結果の直径は δ^* より小さい場合，δ^* を新しい直径に更新する．
10-12 行目:16 分割木の各ノードのキーワード分布ビットマップを利用して，それぞれのキーワード w_i に応じて，子ノードにキーワードを含むことによりリスト $L_i(i = 1, 2, .., m)$ を作る．
14-47 行目:while ループでリスト L_1〜L_m に対して深さ優先で要素の組み合わせを作り，バックトラッキングで枝刈りを行なう．一つのノードセットができれば，木の下へたどる．つまり子ノードのセットを入力として再帰的に $mCKSearch$ を呼び出す (40 行目)．
具体的な例を挙げる．図 7 のように示す．

$L_A = \{n_1, n_2\}$
$L_B = \{n_1, n_6\}$
$L_C = \{n_2, n_6\}$

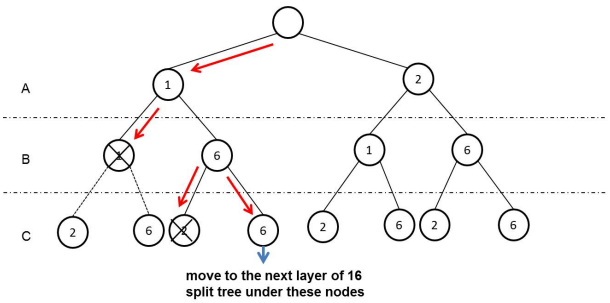


図 7　ノードの組み合わせ

$1, 2, 6$ はノード ID である．もし，L_A の n_1 と L_B の n_1 はルール 1 を満たすなら，L_A の n_1 へ戻り L_B の n_6 を試して、成功する．そして，L_C の n_2 へたどって、L_C の n_2 と L_A の n_1，L_C の n_2 と L_B の n_6 両方を判断して，ルール 1 を満足するので，L_B の n_6 へ戻り L_C の n_6 を試して、成功する．そして，ノードセットを作って，入力として再帰的に $mCKSearch$ を呼び出す．
18-25 行目:ルール 1 が判定される．
28-39 行目:ルール 2 が判定される．

4.4　アルゴリズムの最適化

ルール 1 はキーワードに関連するノードの間の $MINDIST$ を計算して枝刈りを行うが，隣接するノードの間の $MINDIST$ は 0 であるので，枝刈りの能力が低い．枝刈りの能力を高めるため，近似キーワード MBR という概念を提出する．

定義 6(キーワード近似 MBR):ノード n において，n の子ノードのうち，キーワード w_i を含むノード集合から成る MBR を，w_i の近似 MBR と呼ぶ．(n の w_i についての w_i の近似 MBR

は n が持つ w_i のキーワード分布ビットマップから求める)

例えば，図 8 は図 2(a) のノード 1 と 2 におけるキーワード A と C の近似 MBR を示す．左のノード 1 はキーワード A の分布ビットマップを使って，キーワード A はノード 1 の 7,13 番目の子ノードに含まれるので，A に対してノード 1 の近似 MBR は赤い太い線で囲んだ矩形により表示される．同様に，右のノード 2 のキーワード C の近似 MBR は青い線で囲んだ矩形により表示される．ノード間の MINDIST の計算ではなく，代わりに近似 MBR 間の MINDIST を利用して，ルール 1 を判断する最適化方法を提案する．そうすると，隣接ノードであっても，最小距離は 0 ではない可能性があるので，枝刈りすれば，探索空間を減少することができる．

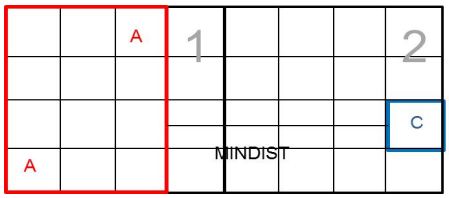


図 8　キーワード近似 MBR

5. 実　　験

Java 言語で図 2 に従った 16 分割木のデータ構造と mCK 検索のアルゴリズムを実装した．実験用マシンは Intel(R) Xeon(R) 2.67GHz の CPU と 12GB のメモリを用いる．OS は Vine Linux5.1 である．対象になるデータは人工生成されることと実際のデータ二つ種類で，提案したデータ構造と mCK 検索のアルゴリズムの検索性能を見る．

5.1 人工生成データ

性能評価について，クエリのキーワードの数 と データの数 (レコード数) の二つのパラメータを変えて検索性能を評価する．最後はキーワードの分布が偏る場合について結果の直径と実行時間の関係を調べる．

5.1.1 クエリのキーワードの数の影響

対象データについて，キーワードの総数が 100 と決めて，各キーワードに 100 個のレコードを関連づけ，ランダムで座標を生成して木に入れる．つまり，レコードの数が 10000 である．16 分割木のセルの容量は 100 とする．実験の結果を図 9 に示す．図 9(b) はランダムで m 個のキーワードを選んで検索を行うのを 1 回として 100 回実行の平均時間である，図 9(a) はノード組み合わせの数は根ノードから葉ノードまでチェックしたノードセットの個数の平均値である．

以下の 3 通りの枝刈り戦略を用いた方式について評価した:

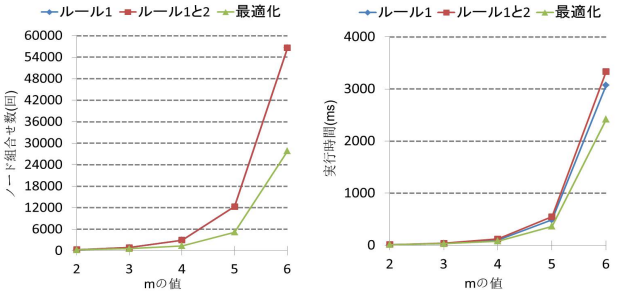
ルール 1　:(単にルール 1 を使う方式である.)
ルール 1 と 2　:(ルール 1,2 両方を使う方式である.)
最適化　:(キーワード近似 MBR を使って $MINDIST$ を求め，ルール 1 と 2 も使う方式である.)

図 9 によると，m の増大に応じて，実行時間とノード組み合

Algorithm 1 mCKSearch($keywordSet, nodeSet, Tree$)

Input: $keywordSet$: 現在に行うキーワードの集合
$nodeSet$: 現在に行うノードの集合
$Tree$:16 分割木
Output: δ^* : 今まで計算した直径の最小値.

```
1: if nodeSet のすべてのノードは葉ノード then
2:   d ← ExhaustiveSearch(nodeSet)
3:   if d < δ* then
4:     δ* ← d
5:   end if
6: else
7:   num ← keywordSet の中にキーワードの数
8:   newNodeSet ← ∅
9:   maxMinDist ← 0
10:  for node i in nodeSet do
11:    L_i ← traverse(i)
12:  end for
13:  k ← 1
14:  while k ≧ 1 do
15:    // 順番で L_k を要素をアクセス
16:    if L_k に次の要素がある then
17:      n ← L_k の次の要素
18:      for node j in newNodeSet do
19:        dist ← MINDIST(j, n)
20:        if dist > δ* then
21:          14 行へ
22:        else if dist > maxMinDist then
23:          maxMinDist ← dist
24:        end if
25:      end for
26:      n を newNodeSet に入れる
27:      if k = num then
28:        for node a in newNodeSet do
29:          flag ← TRUE
30:          for node b in newNodeSet do
31:            if MAXDIST(a, b) ≦ maxMinDist then
32:              flag ← FALSE
33:            end if
34:          end for
35:          if flag = TRUE then
36:            a の keyword を keywordSet から一時的に除く
37:            a を newNodeSet から一時的に除く
38:          end if
39:        end for
40:        mCKSearch(keywordSet, newNodeSet, Tree)
41:      else
42:        k ← k + 1
43:      end if
44:    else
45:      k ← k − 1
46:    end if
47:  end while
48: end if
```



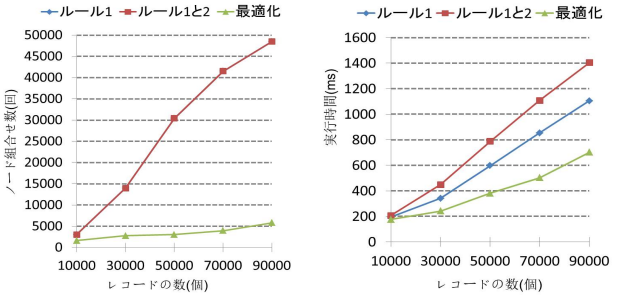
(a) ノード組み合わせの数　　(b) 実行時間

図 9　クエリのキーワードの数の影響

わせの数は指数で増加する．ルール 1 と 2 を使う方式のは単にルール 1 を使う方式より実行時間が多い．原因はデータは地図に均一分布しているから，結果の直径は小さい．そのため，隣接のノードの組み合わせが多いので，ルール 2 は枝刈りできないのみならず，余計な判断時間がかかる．

5.1.2　レコードの数の影響

対象になるデータについて，キーワードの総数が 100，各キーワードに 100,300,500,700,900 のレコードを関連づけて実験を行う．つまり，レコードの総数は 10000,30000,50000,70000,90000 である．16 分割木のセルの容量は 100 とする．クエリのキーワードの数 $m = 4$ とする．実行時間とノード組み合わせの数も 100 回実行の平均値である．結果を図 10 に示す．(図 9(a), 図 10(a) でルール 1 のみの方式はルール 2 の場合とほぼ一致している)



(a) ノード組み合わせの数　　(b) 実行時間

図 10　レコードの数の影響 (m=4)

図 10 によると，レコードの増大に応じて，実行時間とノード組み合わせの数が増加する．16 分割木の階層が増加する と各葉ノードにレコードの数が増えるのも実行時間が増加する原因となる．このテストでは対象データはランダムでデータを生成したため，答えになるオブジェクト集合が近くにある可能性が高い．そこで，閾値 δ^* は速く減少して，大部分の探索空間が枝刈りした場合の結果になっている．

5.1.3　データの偏りの影響

現実のデータは偏る場合もある．つまり，閾値 δ^* が大きく，大部分の探索空間を走査してから，結果を求める場合もある．そこで，偏るデータを生成して行う．生成データの分布を図 11 に示す．

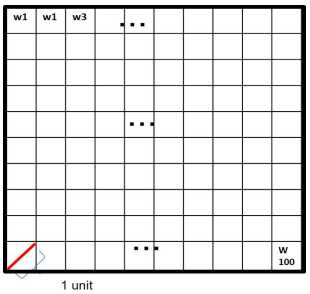
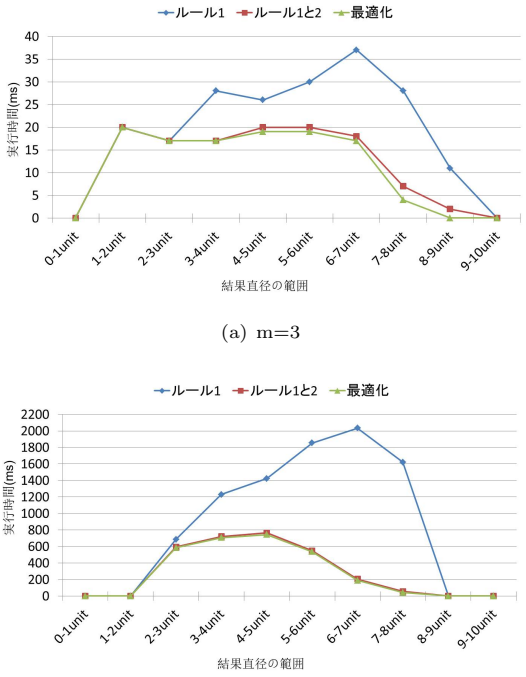


図 11　偏りデータ分布

全空間を 100 セルを区切り，一つのセルにただ一つのキーワードに関連するレコードを入れる．そして，ランダムで $m = 3, 4$ のキーワードのクエリで実行する．一つのセルの対角距離を 1unit として，実行結果の直径 δ を 10 個の間隔 (0-1unit,1-2unit,...,8-9unit,9-10unit) に分けると，δ は必ず 1 つの間隔に落ちる．150 回の実験を行って，δ を分けて，各間隔に落ちた時のクエリ 1 回あたりの実行時間の平均値を図 12 に示す．(図 12(a) が $m = 3$, 図 12(b) が $m = 4$)



(a) m=3

(b) m=4

図 12　結果の直径 δ の影響

図 12 によると，結果の直径とともに実行時間は変化する．データが離れる場合，ルール 2 は明らかに有効に働いている．データが離れて，判断必要なキーワードを削減するので，ノードの組み合わせあるいは葉ノードのデータの組み合わせが減少されるので，検索効率が高くなる．

5.2　実際のデータ

本稿はアメリカの国勢調査局の TIGER(Topologically Integrated Geographic Encoding and Referencing)[8] システムからテキサス州のランドマークデータを取得して，16 分割木のデータ構造を作って，mCK 検索の実験を行った．データ総数

は 6613 個であり，キーワードの数は 35 である．16 分割木のセルの容量は 30 とする．木の高さは 6 である．

性能評価について，人工生成データと同じく，クエリのキーワードの数により検索性能を評価する．ランダムで m 個のキーワードを選んで検索を行うことをテスト 1 回として，各 m に対して 100 回テストを行い，実行時間の平均値を計算した．結果を図 13 に示す．

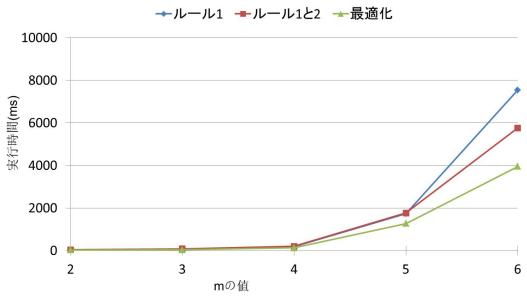


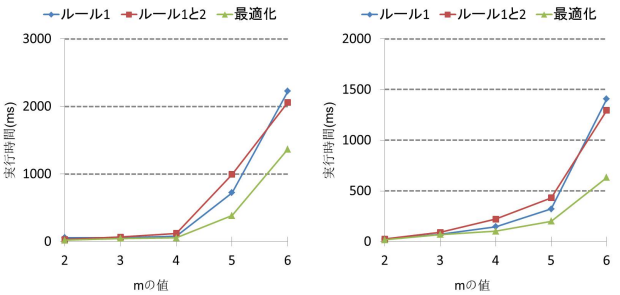
図 13　クエリのキーワードの数 m の影響 (実際のデータ)

図 13 を見ると，ルール 2 はそれほど有効に働いていない．原因は結果の直径は小さい，つまり求めたオブジェクトがお互いに近いからである．また，クエリのキーワードの数が多い場合，実行効率は良くないと判る．原因として，提案のアルゴリズムは順番でノードを組み合わせているため，枝刈り能力がデータの分布より決まる．つまり，結果を見つける時点が遅ければ遅いほど閾値 δ^* の収束が遅く，探索空間木にチェックしたノードセットが多いので，時間がかかると考えられる．

上記の原因を検証するために，δ^* の初期値を指定する場合について同じの実験を行った．

テキサス州の一辺の長さは約 1000km ぐらいで，δ^* の初期値を 100km と 300km 二つ種類を設定する．つまり，直径が 100km と 300km 以内の m 個のキーワードに関連するオブジェクトの集合を求めることになる．

実行結果を図 14 に示す．



(a) δ^* の初期値=300km　　(b) δ^* の初期値=100km

図 14　δ^* の初期値を指定する場合, クエリのキーワード m の数の影響

図 14 によって，δ^* の初期値が小さい場合，実行時間が明らかに減少することができる．そこで，もし値の小さな初期値を求めることができれば，検索効率が高くなるとわかる．この点が今後の課題である．

6. おわりに

本稿では，空間データの mCK 検索問題を扱って，データの更新とデータセットのオーバラップ効率を考えて 2^n 分割木のデータ構造を用いて，それぞれのキーワードに関連するノードの組み合わせ探索を行い，二つの枝刈りルールを利用して効率化を図る方法を提案した．またキーワード近似 MBR を使ってアルゴリズムを最適化した．実験では，均一分布と偏り分布の人工生成データを使って，クエリのキーワードの数とレコードの数の変化に応じる実行時間の変化を調べた．その結果枝刈りルール 1 と 2 と最適化方法は探索空間を減少できることが判った．また，実データで実験を行った．結果によって，クエリのキーワードの数が多い場合，探索効率は良くないが，値の小さな δ^* の初期値を指定することで，探索時間が明らかに減少できると判った．

今後の課題としては，今のアルゴリズムは順番でノードのセットを組み合わせるので，枝刈りの効率はデータ分布による異なる．そのため，探索順を priority 順に変えることや δ^* の上限を早く決める方法を導入することが重要である．

謝辞：本研究の一部は科研基盤 C24500109 の助成による．

文　献

[1] Christian S. Jensen. “Data Management on Spatial Web”,keynote, Proceedings of the VLDB Endowment, Vol. 5, No.12, 2012
[2] R. Hariharan, B. Hore, C. Li, and S. Mehrotra. “Processing spatial- keyword queries in geographic information retrieval systems.“ In SSDBM, pp. 16-25, 2007.
[3] Z. S. Li , B. H. Zhen , W. C. Lee, D. L. Lee , X. F. Wang “ IR-Tree: An Efficient Index for Geographic Document Search” IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 23, NO. 4, APRIL 2011, pp. 585-599
[4] D.X.Zhang, Y.M.Chee, Anirban Mondal, Anthony K.H.Tung, Masaru Kitsuregawa, ”Keyword Search in Spatial Databases: Towards Searching by Document,” IEEE ICDE, pp.688-699, 2009.
[5] http://research.microsoft.com/en-us/projects/geolife/
[6] http://twitter.com/
[7] D. Papadias, N. Mamoulis, and Y. Theodoridis. “Processing and opti- mization of multiway spatial joins using R-trees.” Proc. PODS, pp. 44—55, 1999.
[8] http://www.census.gov/geo/maps-data/data/tiger.html