DEIM Forum 2012 D8-4

特化したR木空間索引を用いた効率的な逆最遠傍検索

劉　健全† 陳　漢雄† 北川　博之†,††

† 筑波大学大学院システム情報工学研究科 〒305–8577 茨城県つくば市天王台 1–1–1
†† 筑波大学計算科学研究センター 〒305–8577 茨城県つくば市天王台 1–1–1
E-mail: †ljq@dblab.is.tsukuba.ac.jp, ††{chx,kitagawa}@cs.tsukuba.ac.jp

あらまし 本論文では，逆最遠傍 (Reverse Furthest Neighbors, RFN) 探索問題に対して，R 木構造を基に特化した空間索引と，これを用いた検索アルゴリズムを提案する．先行研究では，従来手法の高い計算量を大幅に削減でき，任意位置のクエリ q に適用可能な検索手法を提案したが, 効率的に十分ではなく，多次元データにも適応しにくいという問題がある．本研究では，ピボットセットを選出して，同一ピボットを最遠傍とするデータをまとめて R 木に格納し，最遠傍距離を埋め込めるように特化した空間索引，およびこの空間索引を用いた RFN 検索アルゴリズムを提案する．さらに、多次元への拡張を検討し，合成データと実データを用いて提案手法の効率とスケーラビリティを検証する．

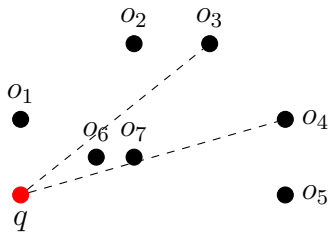
キーワード 類似検索，逆最遠傍 (RFN)，R 木，空間索引

1. はじめに

過去十年間，類似検索の科学研究・開発への応用が急増している．例として，パターン認識や [1], [2]，画像検索 [3]，時系列マッチング [4] などが挙げられる．これらの応用に伴って，類似検索に関する研究は活発にされている．類似検索に関する問い合わせ処理は，k 近傍 (k-Nearest Neighbors, k-NN) や，範囲問い合わせ (Range Query)，逆最近傍 (Reverse Nearest Neighbors) などの変形がある．これらに対するオンライン問い合わせ処理の要求が急増しており，様々な処理技術が提案された．それらは主に，空間分割索引法 (例：k-d-b-tree [5])，データ分割索引法 (例：R-tree [6])，連続スキャン法 (例：VA-file [7]) との三種類に分けられる．

しかし，近傍の反対，すなわち，遠傍に対する問い合わせ処理はこれまで今後の課題とされ，特に逆最遠傍 (Reverse Furthest Neighbors, RFN) の問い合わせ処理に関する集中な議論は未着手となっていた．しかし，実世界の応用では，効率的な逆最遠傍の問い合わせ処理は有意義である．Yao たち [8] は，初めて逆最遠傍問題に関する正式な定義を示し，R-tree に基づく問い合わせ処理方法をともに提案した．その後，我々がピボットの選出によるフィルタリングとリファインメントに基づいて，効率を向上させた問い合わせ処理方法を提案した [9]〜[11]．あるデータセット O と任意のクエリ q が与えられたとき，O の中から q を最遠傍とする全てのオブジェクトの集合を，q の逆最遠傍 (RFN) と言う．また，この問題を解くことを逆最遠傍検索と言う．

例として，図 1 を用いて説明する．図 1 では，RFN の問い合わせ処理を示している．オブジェクト集合 O 及びクエリ q が与えられたとき，q の逆最遠傍は集合 $\{o_3, o_4\}$ になる．単純な方法でこれを求めるには，各オブジェクト o_i において，o_i と q の距離，及び o_i と他の $o_j (i \neq j)$ の距離を算出し，o_i は q を最遠傍とするかどうかを確認すればよい．図 1 の例では，o_3 と o_4 のみが q をそれぞれの最遠傍 (それぞれの距離は破線で表す) とするので，q の逆最遠傍の解になることが分かる．



$$O = \{o_1, o_2, \cdots, o_7\}$$

図 1 RFN の問い合わせ例

逆最遠傍検索の応用について，多くの科学研究分野で指摘された [12]〜[17]．伝統的な応用を言えば，最適な位置で新しい施設を配置するような施設配置に関する最適化問題は，日常生活にとって非常に重要であると述べられた [13], [16]．新しい施設として，例えば，ごみ捨て場や，化学工場，スーパー，コンビニ，ホテル，警察署などが挙げられる．施設配置に関する最適化問題の中，特に，競合的な施設配置問題は

まだ開放問題の一つであると示された [15]．このような応用問題を解決するには，逆最遠傍検索を効率的に行うことが非常に役立つと考えられる．

次に，逆最遠傍問題の実用性を示すため，我々の先行研究 [9]〜[11] に基づき，代表の一つとして，以下に具体的な応用例を示す．

応用例：競合的な施設配置問題を考える．あるコンビニのチェーンストア会社 (以下，7-Eleven) が新しいコンビニを一つ開設する計画を立て，候補位置から最適な位置を選択するような実問題として考えてみる．7-Eleven 店を新設する計画の段階には，会社経営者の立場で考慮すると，利益を最大の要因として考えられる．このため，新設位置の周辺にいる潜在的な顧客の数量や，新設位置の近くに既存する 7-Eleven 店との競合性，所在地の交通の利便性などは重要なファクターとみなせる．この中，既存支店との競合性を考えると，既にある店の利益を損なうのをあらかじめ避ける必要がある．よって，位置の視点で図れば，新設位置が既存位置から離れるほどよいと考えられる．しかし，全ての既存支店とも最も離れるのが現実ではないため，妥協案として，最も多くの既存支店が新規したい店の位置を最遠とみなしたほうが合理であるだろう．この際に，逆最遠傍問題を解くことにより，最適な位置を選ぶことができる．例えば，新設位置の候補集合を $Q = \{q_1, q_2, \cdots, q_m\}$ とし，既存支店の位置集合を $O = \{o_1, o_2, \cdots, o_n\}$ とする．前述した考え方にしたがって，逆最遠傍検索を行いながら，$arg \max_{q_i \in Q}\{|RFN(q_i)$ の解 $|\}$ の結果を求めて，最適な q_i を選出できる．この q_i は最も多くの既存支店の最遠傍となる最適な位置である．すなわち，選出された q_i は既存する店に利益上の影響を与えることが最も少ないと考えられる．

先行研究 [9]〜[11] では，従来手法の高い計算量を大幅に削減でき，任意位置のクエリ q に適用可能な検索手法を提案したが, 効率的に十分ではなく，多次元データにも適応しにくいという問題がある．そこで，本研究では，ピボットセットを選出して，同一ピボットを最遠傍とするデータをまとめて R-tree に格納し，最遠傍距離を埋め込めるように特化した空間索引，およびこの空間索引を用いた RFN 検索アルゴリズムを提案する．さらに、多次元への拡張を検討し，合成データと実データを用いて提案手法の効率とスケーラビリティを検証する．実験結果によって，本稿で提案するアプローチは，関連研究及び先行研究で提案された検索手法より，検索効率をさらに大幅に向上させることが分かった．

本論文の構成は以下の通りである．第 2 章で関連研究及び先行研究を紹介する．第 3 章では，本研究の提案手法に関わる空間索引構造と検索での扱い方を詳しく説明する．続いて，新たな検索アルゴリズムを提案し，第 4 章で評価実験によって提案手法が効率向上であることとスケーラビリティを示す．最後に，簡単なまとめと，今後の課題を述べる．

2. 関連研究と先行研究

類似検索に関する技術はこれまでにもよく研究されていた．本研究と最も関連のある問い合わせは主に 3 つのタイプに分けられる．すなわち，(k-) 最近傍検索 (NN, あるいは k-NN)，逆 (k-) 最近傍検索 (RNN, あるいは RkNN)，逆最遠傍 (RFN) である．

最近傍検索 については，早期に R-tree [6] を用いる深さ優先探索法 [18], [19] と最良優先探索法 [20] が提案された．近年，Athitsos たちは距離に基づくハッシュ手法を用いた近似最近傍検索を提案した [21]．また，Tao たちは LSB-tree による高次元データにおける最近傍検索を提案した [22]．Chen たちは関数索引を初めて提案し，高次元データにおける連続スキャン法の検索効率を向上させた [23], [24]．

逆最近傍検索 に関しては，Korn と Muthukrishnan が初めて逆最近傍の問い合わせを提案した [25]．それ以降，様々な問い合わせ処理手法が提案された．例えば，RdNN-Tree [26], TPL [27]，MRkNNCop-Tree [28] などがある．これらには，大量なデータに対する RNN 応用例も示されている．

逆最遠傍検索 について，本研究と最も関連するものは [8] である．Yao たちは初めて逆最遠傍検索 RFN を定義した．RFN の問い合わせを処理するため，彼らは Progressive Furthest Cell (**PFC**) と Convex Hull Furthest Cell (**CHFC**) というアルゴリズムを提案した．これらのアルゴリズムでは，R-tree とともに，最遠ボロノイセル (furthest Voronoi cell, fvc) を利用して，あるデータオブジェクト $o \in O$ が q の RFN になるかどうかを判断する．ここで，$fvc(q, O)$ は，データセット O における，クエリ q に関する凸多辺形を表す．$fvc(q, O)$ の領域は，q と $o(o \in O)$ の全てのペアの垂直二等分線を引いて，データ空間を 2 つの部分空間に分けて，q と離れている部分空間をそれぞれの交差を取ったものがクエリ q に関する凸多辺形領域になる．この領域が q の逆最遠傍になる $o \in O$ を含むことが証明されている．この性質を利用するため，**PFC** アルゴリズムでは，R-tree における走査をしながら，$fvc(q, O)$ を構築する．同時に，$fvc(q, O)$ を用いて，各 MBR あるいはデータオブジェクトが解になるかをチェックする．しかし，このアルゴリズムでは，解にならないものはそのまま捨てられない．解にならないものを用いて，より狭い $fvc(q, O)$ 領域を再構築することは可能であるため，この後処理は非常に非効率的である．**PFC** を改善するため，彼らはさらに **CHFC** アルゴリズムを提案した．**CHFC** では，逆最遠傍問題に関する特有な凸包の性質を示し，この性質に着目して $fvc(q, O)$ を構築することで，より効率のよい検索が可能になる．**CHFC** アルゴリズムは **PFC** より速いが，効率の悪い処理がまだ含まれている． 例えば，凸包を計算することと，fvc という不規則な領域の計算，および R-tree を用いた fvc が包含するデータのチェックを行うこと

は計算コストが高い．

これらの高価な計算コストを削減するため，我々の先行研究では新たなアプローチを提案した．先行研究 [9] において効率向上のための目標は，1) 凸包の計算を避けること，2)R-tree を使った包含チェックの計算を避けること，3) 検索するときに高価な距離計算をなるべく削減すること，であった．先行研究 [9] では，データセットに関する凸包を前計算し，ピボットセットとして選出する．さらに，前計算でピボットセットとデータセットとのペア毎の距離索引をあらかじめ作成しておき，同時にデータセットにある各オブジェクトとそれの最遠傍距離を索引表で記録しておく．[9] で提案した **PIV** アルゴリズムでは，距離索引を用いて，三角不等式によるフィルタリングとリファインメントを行う効率的な連続スキャン法であった．**PIV** 法は Yao たちの従来手法より計算の高速化が実現できたが，任意位置，特にデータセットに遠く離れたクエリに有効ではないという問題があったため，先行研究 [10], [11] では，**PIV** の改善案として **PIV**$^+$ アルゴリズムを提案した．**PIV**$^+$ 法では，データセット内の全てのオブジェクトが逆最遠傍の解になるようなクエリの “安全区域” を発見した．この安全区域に入ってくるクエリに適応可能なフィルタリング条件の定義及び理論証明を行い，効率をさらに向上させたアルゴリズム **PIV**$^+$ を提案した．

しかし，関連研究と先行研究で提案されたアルゴリズムの複雑さを表 1 で簡単にまとめてみると，今までのアプローチは検索効率に対して十分に向上させたことではないことが分かる．関連研究で提案した **CHFC** アルゴリズムは，従来の単純な探索法 (Brute-Force Search: **BFS**) でかかる N の 2 乗の複雑さから，線形程度の複雑さへの削減ができた．我々の先行研究では，**PIV** アルゴリズムと **PIV**$^+$ アルゴリズムとも線形複雑さへの削減ができて，**PIV**$^+$ が **PIV** よりさらに優位であったが，本研究では，ログオーダに削減できるようなアプローチを目指している．

表 1　既存手法の複雑さの比較

手法	最良	平均	最悪
PIV [9]	$O(m)$	$O(\alpha_1 \cdot N)$	$O(m \cdot N)$
PIV$^+$ [11]	$O(m)$	$O(\alpha_2 \cdot N)$	$O(m \cdot N)$
CHFC [8]	$O(m^2+\log_M N)$	N/A	$O(N+m^2+m \cdot M^{h+1})$
BFS [8]	$O(N^2)$	$O(N^2)$	$O(N^2)$

m: ピボットの数; α_1, α_2: 線形比率 (通常は 1 より小さい，かつ $\alpha_2 < \alpha_1$);
h: R-tree の高さ; M: R-tree のノード容量; N: データの数;

3. 提案手法

本章では，RFN 検索の計算複雑さをログオーダに削減することを目的とし，基本アイディアと具体提案を紹介する．まず，基本的な考え方としては，先行研究で用いられたピボットセットを利用して，ピボットに関わる距離索引を埋め込めるように特化した R-tree の空間索引を提案する．この空間索引を利用し，検索効率は木構造の探索によるログオーダへ削減する．理論上 $O(m \log_M \frac{N}{m})$ への削減が可能となる．次に特化した R-tree のデータ構造について説明する．

3.1 特化した R 木構造：CHRTree

本節では，図 2 にある 2 次元データを例として，これらのデータを索引する具体なデータ構造，および索引を構築する流れを詳しく述べる．

まず，関連研究 [8] で解明された性質，および先行研究 [9]〜[11] で利用していたピボットセットについて簡単にまとめて説明する．

［**定理** 1］　データセット O 及びその凸包 C_O に対して，任意のクエリ q が与えられたとき，q の RFN の解が空ではないための必要十分条件は，q が凸包 C_O の外側あるいは境界線にあることである．

定理 1 により，与えられた q の位置によって，RFN 検索のフィルタリングをすることができる．すなわち，q が凸包の内部にあると判断できる時点で，RFN 検索の解は存在しないことが判断できるので，距離計算などの処理を行う必要はない．

また，データセットを単体として，定理 1 から，データセットにある任意のオブジェクト o を q としてみなすと，RFN の解が存在し得るのは，凸包の頂点であることが分かる．例えば，図 2 にあるデータセット $O = \{o_1, o_2, \cdots, o_{18}\}$ では，$o_{11}, o_{17}, o_8, o_{10}$ を順序に赤い実線で連結している凸四辺形が凸包となる．定理 1 によると，データセットのオブジェクトを 1 つずつ q とみなしても，凸包の頂点であるオブジェクト $o_{11}, o_{17}, o_8, o_{10}$ のみ RFN の空ではない解が存在し得る．図 2 では，$o_{11}, o_{17}, o_8, o_{10}$ の RFN の解をそれぞれ異なる色の矩形で囲んで示している．例えば，o_{10} の RFN の解は向こう側の $o_{15}, o_{16}, o_{17}, o_{18}$ である．

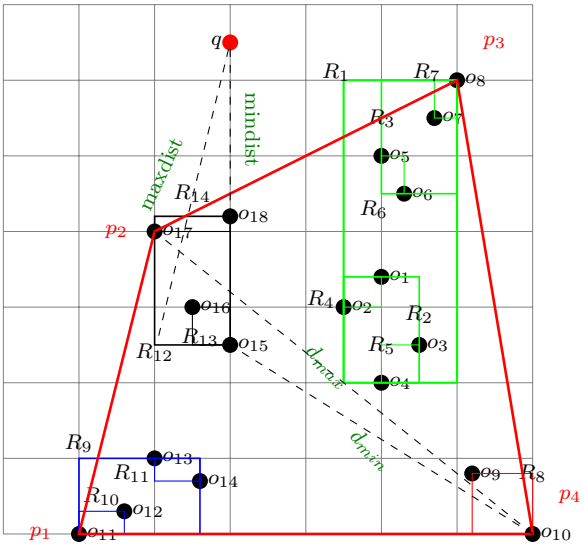


図 2　索引される 2 次元データとクエリ q

定理 1 に基づいて，先行研究では，凸包の頂点をピボットとして選出する．例えば，図 2 にある $o_{11}, o_{17}, o_8, o_{10}$ をそれ

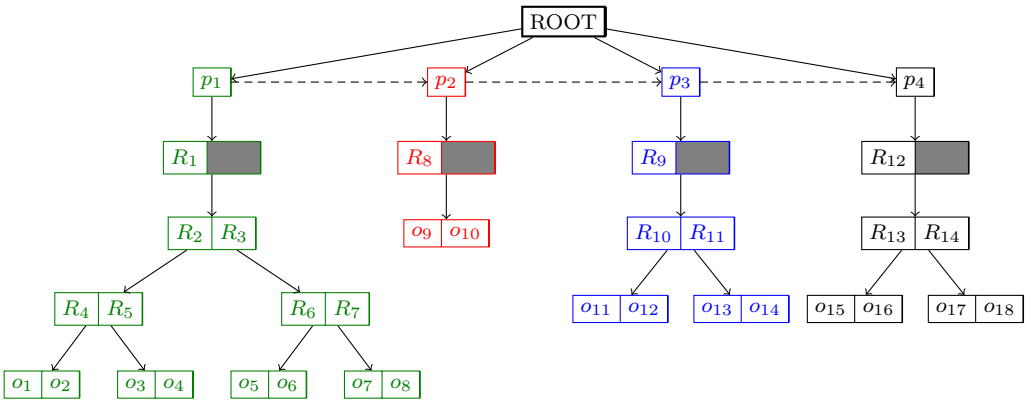


図 3　特化した R 木空間索引 (**CHRTree**)

ぞれピボットとして，ピボットセット $S_{piv} = \{p_1, p_2, p_3, p_4\}$ へ選出する．さらに，データセットにある各オブジェクト o の最遠傍となるピボットとの距離 $\max_{p_i \in S_{piv}} \{dist(o, p_i)\}$ を前計算して距離索引を作成する．本研究では，ピボットセット S_{piv} と距離索引を続けて利用するが，距離を格納する方法を変えて，木構造に埋め込めるようにする．

次に，S_{piv} の扱い方と R 木の特化し方について説明する．先行研究では，ピボットと三角不等式を合わせて検索のフィルタリングで利用していたが，本研究での扱い方は異なる．本研究では，$\forall o \in O$ について，ピボットを最遠傍とする距離 $\max_{p_i \in S_{piv}} \{dist(o, p_i)\}$ を前計算する同時に，同じピボットを最遠傍とするオブジェクトを同一グループにまとめる．そして，それぞれのグループにあるデータを特化される R-tree で索引する．例えば，図 2 にあるすべてのデータはピボットによる 4 つのグループに分けられて，異なる色の最小包含矩形 (MBR) で囲んで示し，図 3 で索引された独自の木構造に同じ色で表している．具体に，p_1, p_2, p_3, p_4 をそれぞれ最遠傍とするデータグループは，緑，赤，青，黒の色で区別している．それぞれは，1 つの R-tree として，データ索引を行う．その後に，対応するピボット $p_i, i = 1 \cdots 4$ をそれぞれの R-tree のルートとして木へ加える．また，4 つの R-tree をサブ木として，共通ルートに付けて，最終的に大きな木を 1 つとなるような独自の空間索引構造になる．さらに，各サブ木のルートへアクセスしやすいため，凸包の頂点順にリンクさせる．ノードの容量を 2 に仮定して，全体的には，図 3 で示すように特別な R 木構造である．本稿では，このように特化した R 木空間索引を **CHRTree** と言う．**CHRTree** は Convex Hull (凸包,CH) を用いて特化した R-tree という意味の略称である．

また，**CHRTree** を利用して RFN 検索を効率に行うため，**CHRTree** にあるサブ木の R-tree でのノードに保存する情報をさらに特化する必要がある．同一ピボットによるグループされたデータの R-tree 索引を構築する際に，ノードのエントリを次のように特化する．

a）葉ノードであるときに

各エントリ (e) には次の情報を保持する．

- データへのアクセス・ポインター．
- データとその最遠傍との距離，すなわち，データとサブ木のルートにあるピボットとの距離 $\max_{p_i \in S_{piv}} \{dist(o, p_i)\}$, $e.d_{max}$ で表す．

b）中間ノードであるときに

各エントリ (e) は MBR となり，MBR に関して次の情報を保存する．

- 子ノードへのアクセス・ポインター．
- 子ノードを囲む MBR の座標情報．
- 子ノードにある全てのエントリ (child c) の d_{max} の最小値，$e.d_{min}$ で表す．すなわち，$e.d_{min} = \min_{c \in e} \{c.d_{max}\}$．
- 子ノードにある全てのエントリ (child c) の d_{max} の最大値，$e.d_{max}$ で表す．すなわち，$e.d_{max} = \max_{c \in e} \{c.d_{max}\}$．

図 2 を用いて説明すると，例えば，中間ノードにある R_{12} に対応するエントリの d_{max} と d_{min} を破線で示している．$R_{12}.d_{max}$ は o_{17} とピボット p_4 との距離 $dist(o_{17}, p_4)$ であり，$R_{12}.d_{min}$ は o_{15} と p_4 との距離 $dist(o_{15}, p_4)$ である．

a) と b) を統合して考えれば，サブ木にある任意の 1 つエントリ (e) に対して，$e.d_{max}$ を意味するのは，エントリ e が索引している全てのデータの中に必ずある 1 つのデータが対応するピボットを最遠傍とし，その距離は $e.d_{max}$ であること．すなわち，対応するピボットを最遠傍とする距離の上界である．もし，クエリ q とピボットに関連するオブジェクト o との距離はこの上界より大きければ，o は必ず q の逆最遠傍になると判断できる．一方，$e.d_{min}$ については，$e.d_{max}$ の逆意味である下界を表す．すなわち，この下界距離より小さければ，ピボットに対応するオブジェクトは q の逆最遠傍にならないと判断できる．

よって，$e.d_{max}$ を利用したうえ，Roussopoulos ら [18] が開発したクエリ q と MBR(e) とのバウンディング (bounding) 距離を併用すれば，**CHRTree** による効率的な枝刈りルールを設計することが可能となる．次に **CHRTree** における木

探索の枝刈りルールを詳細に述べる．

3.2 CHRTree における枝刈り

本節では，クエリ q と索引木にある MBR とのバウンディング距離 MINDIST と MAXDIST を定義し，これらに基づく木探索の枝刈りルールの設計を行う．

まず，D 次元の長方形 MBR(e) の左下角の座標を $(s_1, s_2, \cdots, s_D)$ で，右上角の座標を $(t_1, t_2, \cdots, t_D)$ で表す．D 次元のクエリ q の座標を $(x_1, x_2, \cdots, x_D)$ で表す．そして，距離はユークリッド距離を使い，MINDIST と MAXDIST を次のとおりに定義する．

［**定義** 1］（MINDIST） クエリ q と MBR(e) との最小距離，$mindist(q, e)$，は次の式で定義する．

$$mindist(q, e) = \sqrt{\sum_{j=1}^{D} |x_j - r_j|^2},$$

where:

$$r_j = \begin{cases} s_j, & x_j < s_j \\ t_j, & x_j > t_j \\ x_j, & otherwise \end{cases}$$

［**定義** 2］（MAXDIST） クエリ q と MBR(e) との最大距離，$maxdist(q, e)$，は q から MBR と最も遠い頂点との距離を最大距離という．次の式で定義する．

$$maxdist(q, e) = \sqrt{\sum_{j=1}^{D} |x_j - r_j|^2},$$

where:

$$r_j = \begin{cases} t_j, & x_j \leqq \frac{s_j + t_j}{2} \\ s_j, & otherwise \end{cases}$$

次に，MINDIST と MAXDIST と合わせて，d_{max} および d_{min} を利用して，**CHRTree** における木探索の枝刈りルールを設計する．

［**ルール** 1］（上界 1） エントリ (e) がデータポイントである場合，q と e の実距離 $dist(q, e)$ は条件 $dist(q, e) > e.d_{max}$ を満たせば，データ e は q の逆最遠傍の解である．逆の場合には，解ではない．

［**ルール** 2］（上界 2） エントリ (e) が MBR である場合，$mindist(q, e) > e.d_{max}$ が満たせば，e に含まれる全ての子ノード (再帰可) にあるデータは q の逆最遠傍の解になる．

［**ルール** 3］（下界） エントリ (e) が MBR である場合，$maxdist(q, e) \leqq e.d_{min}$ が満たせば，e に含まれる全ての子ノード (再帰可) にあるデータは q の逆最遠傍の解にならない．

上記のルールを用いることにより，e の子ノードに対して再帰的にアクセスすることが必要ないと判断でき，木探索の枝刈りをすることができる．これによって，**CHRTree** における RFN 検索の効率を大幅に向上させることが可能となる．

3.3 検索アルゴリズム

続いて，RFN 検索問題に対し，**CHRTree** 空間索引および前述した枝刈りルールを用いて，効率的な深さ優先検索アルゴリズムを提案する．実際に，RFN 検索を行うには，全てのサブ木を走査することは必要であるので，枝刈りの前後順序のみ異なるが，枝刈りの効果が変わらない．よって，幅優先探索も最良優先探索も用いることが可能だが，理論上の検索効率は変わらないため，深さ優先の探索方式のみでアルゴリズムを設計する．

Algorithm 1: RFNQuery (Query q, **CHRTree** T)

Input: Query q, **CHRTree** T.
Output: Answer set S.

```
1  begin
2      Initialize answer set S ← ∅;
3      Convex hull CH ← T.getConvexHull() ;
4      if q inside CH then
5          return ∅
6      foreach p ∈ CH do
7          Initialize stack L ← ∅ ;
8          subT ← T.getSubTree(p) ;
9          L.push(subT.root) ;
10         while L is not empty do
11             Entry e ← L.pop() ;
12             if e is a point then   // e is a point
13                 if dist(q, e) > e.d_max then
14                     Insert e into S ;
15                 else
16                     Discard e ;
17             else // e is an MBR
18                 if maxdist(q, e) ≦ e.d_min then
19                     Discard e ;
20                 else if mindist(q, e) > e.d_max then
21                     Insert all points o ∈ e into S ;
22                 else
23                     foreach child c of e do
24                         L.push(c) ;
25     return S
```

Algorithm 1 は，二段階に分けて処理を行う．3 行目から 5 行目までは，定理 1 が示唆するフィルタリングを行う．もし q が凸包に入らない場合は，次の段階の処理に入る．3 行目では，getConvexHull() 関数を呼び出し，索引木 T にあるそれぞれのサブ木のルートとなるピボットを取り出して，凸包の構築を行う．6 行目から 24 行目までの段階では，それぞれにピボットに関わるグループデータのサブ木について深さ優先に探索を行う．7 行目では，深さ優先探索のために空スタック L を用意する．8 行目にある getSubTree() を呼び出

し，ピボット p に関するサブ木 $subT$ を取得する．そして，9 行目で $subT$ のルートをスタック L に入れておいて，次の while 循環に繰り返し探索する．13 行目，18 行目，20 行目では，ルール {1,2,3} をそれぞれ順次にチェックを行い，条件を満たす場合は枝刈りをする．23 行目では，エントリ e が指している全ての子ノードをスタック L へ押し入れて，探索を循環させる．最後に，全てのサブ木における探索が終われば，解集合 S を戻り値として返す．

Algorithm 1 の複雑さについて分析すると，理論的に第二段階の for 循環が主なコストを占めるということが分かる．m をピボットセットのサイズ，N をデータセットのサイズ，M を R-tree のノード容量として仮定すると，1 つのピボットに関するサブ木における探索のコストは $O(\log_M \frac{N}{m})$ である．各ピボットにかけ合わせると，提案したアルゴリズムの理論的な複雑さは平均的に $O(m \log_M \frac{N}{m})$ となる．最良の場合は，for 循環内の処理を実行せずに凸包のチェックのみでアルゴリズムが終了なので，複雑さは $O(m)$ となる．また，最悪の場合だと，特に高次元の場合，索引木にある全てのノードをアクセスする必要であるため，線形探索よりも遅い．

また，関連研究および先行研究と比べ，本研究で提案するアルゴリズムは多次元データに対する逆最遠傍検索へ拡張しやすい利点もある．4 行目にある q が凸包に入るかどうかの判断計算を多次元に適応すれば，本アルゴリズムは修正せずすぐに応用できる．ただし，**CHRTree** 索引を構築するときのみ，多次元凸包の計算にかかる前計算コストは高くなることは予想される．**CHRTree** の構築コストを次の式で表すことができる．

$$O(N^{\lfloor \frac{d}{2} \rfloor + 1}) \text{ [29]} + O(N \cdot (m + \log_M \frac{N}{m}))$$

左の部分は d 次元の凸包を計算するにかかる時間であり，右の部分は N 個のデータを m 個のサブ木から探し，木へ挿入するにかかる時間である．よって，多次元の **CHRTree** を構築するには，凸包の計算が主なコストとなることが分かる．

4. 実験と評価

本論文の提案手法の有効性を確認するため，実験を行った．実験環境は以下の通りである．

- CPU: Intel(R) Xeon (R) 2.83 GHz
- 主記憶: 16 GB
- 実装言語: C++

実験データには，2 次元のみの合成データと実データを用いた．合成データはランダムクラスタ分布 (RC) のデータを使った．実データはアメリカの道路ネットワーク図 (Map) を利用した(注1)．

上記の環境で合成データと実データのそれぞれについて，先行研究 [9] の **PIV** 法, [11] の **PIV**$^+$ 法と，関連研究 [8] の検索手法 **CHFC** と，本研究の **CHRTree** を用いたアルゴリズムとの比較実験を行った．フィルタリング率，実行時間 (I/O を含む) と I/O のアクセス回数を比較項目として評価を行った．以下にその結果を示す．

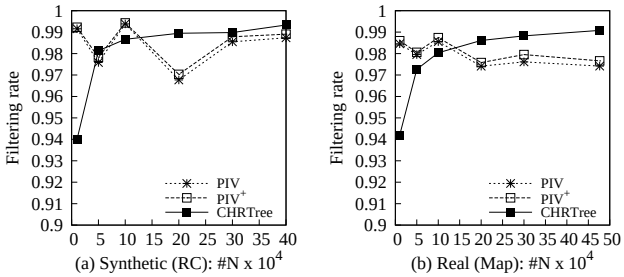


図 4　フィルタリング率：(a) 合成データ，(b) 実データ．

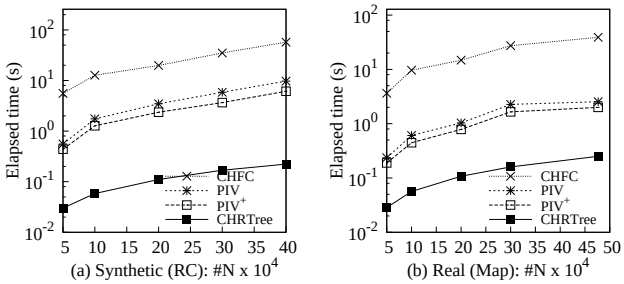


図 5　アルゴリズムの実行時間：(a) 合成データ，(b) 実データ．

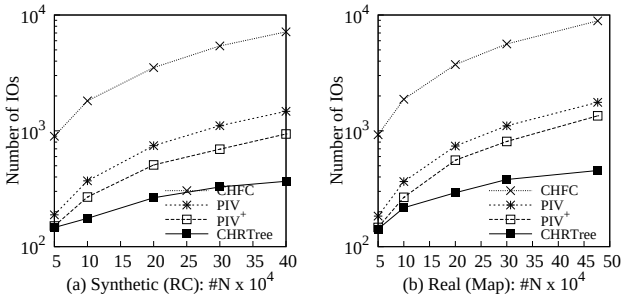


図 6　I/O のアクセス回数：(a) 合成データ，(b) 実データ．

まず，図 4 では，合成データ (RC) および実データ (Map) において，先行研究の **PIV** 法，**PIV**$^+$ 法と本論文の **CHRTree** 法とのフィルタリング率の比較を示す．この図で示す結果から，本研究の提案手法は先行研究の手法よりフィルタリング率が上がることが分かった．理由としては，データセットがピボットによるグループに分けられ，空間索引されるため，先行研究より集中にフィルタリングすることができるからだと考えられる．スケーラビリティに関しては，データセットに関わらずにデータオブジェクトの数 (N) が増えるほど，グループに振り分けられたデータが増加し，グループのデータ密度が上がってくるため，本研究での提案のほうがフィルタリング率が向上する傾向が見られる．

(注1)：http://www.cs.fsu.edu/~lifeifei/SpatialDataset.htm

また，**CHFC** 法，及び **PIV**，**PIV**$^+$ 法とのアルゴリズムの全体の実行時間を比較して，図 5 より本研究の提案手法の効率とスケーラビリティを評価した．図 5 の (a) では，合成データを用い，(b) では実データを用いた．それぞれのデータにおける実行時間を示す．結果によって，逆最遠傍問合せ問題に対して，本提案の **CHRTree** 法が最も速いことが実証された．**CHFC** 法に比べると，処理速度で約 2 桁 (100 倍) の向上が見られた．先行研究よりも，処理速度は 1 桁 (10 倍) 程度向上した．さらに，データセット O のサイズの変化に対し，その他の手法より，本提案手法のほうがより安定的であることが分かった．

さらに，I/O のアクセス回数のみの比較を行い，図 6 で本研究の提案手法の効率を示す．本提案の **CHRTree** 法は先行研究および関連研究の手法より，半分以上の I/O コストを省けることが分かった．

5. ま と め

本論文では，逆最遠傍 (RFN) 検索問題に対して，R-tree 構造を基に特化した新たな空間索引を構築し，これを用いた効率的な検索アルゴリズムを提案した．特に，データ構造の特化によって，RFN 問題に関連する特有な凸包の性質に着目し，最遠傍距離も索引に埋め込むことで，検索効率を大幅に向上させるアルゴリズムを提案・実装した．また，合成データと実データを用いて提案手法の効率とスケーラビリティの検証から，先行研究の手法と従来手法より顕著に効果があることを示した．

今後の課題として，本提案手法は多次元にそのまま適用できるが，多次元凸包の計算量の増加に伴い，索引構築に関連する前計算コストが増える．この問題を改善することは，主要な課題である．また，追加実験として，本提案手法が多次元データに対する逆最遠傍検索にも有効であることを示す計画である．

謝辞 本研究の一部は科学研究費補助金基盤研究 (#21240005) による．

文 献

[1] Serge Belongie, Jitendra Malik, and Jan Puzicha. Shape matching and object recognition using shape contexts. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(4):509–522, 2002.

[2] Kristen Grauman and Trevor Darrell. Fast contour matching using approximate earth mover's distance. In *CVPR (1)*, pages 220–227, 2004.

[3] Ritendra Datta, Dhiraj Joshi, Jia Li, and James Ze Wang. Image retrieval: Ideas, influences, and trends of the new age. *ACM Comput. Surv.*, 40(2), 2008.

[4] Rakesh Agrawal, Christos Faloutsos, and Arun N. Swami. Efficient similarity search in sequence databases. In *FODO*, pages 69–84, 1993.

[5] J. T. Robinson. The k-d-b-tree:a serach structure for large multidimensional dynamic indexs. In *SIGMOD*, pages 10–18, 1981.

[6] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD*, pages 47–57, 1984.

[7] Roger Weber, Hans-Jörg Schek, and Stephen Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *VLDB*, pages 194–205, 1998.

[8] Bin Yao, Feifei Li, and Piyush Kumar. Reverse furthest neighbors in spatial databases. In *ICDE*, pages 664–675, 2009.

[9] Jianquan Liu, Hanxiong Chen, Kazutaka Furuse, and Hiroyuki Kitagawa. An efficient algorithm for reverse furthest neighbors query with metric index. In *DEXA*, pages 437–451, 2010.

[10] Jianquan Liu, Hanxiong Chen, Kazutaka Furuse, Hiroyuki Kitagawa, and Jeffrey Xu Yu. On efficient distance-based similarity search. In *ICDM PhD Forum*, pages 1199–1202, 2011.

[11] Jianquan Liu, Hanxiong Chen, Kazutaka Furuse, and Hiroyuki Kitagawa. An efficient algorithm for arbitrary reverse furthest neighbor queries. In *APWeb*, 2012. (to appear).

[12] Frank Plastria. Static competitive facility location: An overview of optimisation approaches. *European Journal of Operational Research*, 129(3):461–470, 2001.

[13] Trevor S. Hale and Christopher R. Moberg. Location science research: A review. *Annals OR*, 123(1-4):21–35, 2003.

[14] Hee-Kap Ahn, Siu-Wing Cheng, Otfried Cheong, Mordecai J. Golin, and René van Oostrum. Competitive facility location: the voronoi game. *Theor. Comput. Sci.*, 310(1-3):457–467, 2004.

[15] Abdullah Dasci and Gilbert Laporte. A continuous model for multistore competitive location. *Operations Research*, 53(2):263–280, 2005.

[16] Frank Plastria and Lieselot Vanhaverbeke. Discrete models for competitive location with foresight. *Computers & OR*, 35(3):683–700, 2008.

[17] Sergio Cabello, José Miguel Díaz-Báñez, Stefan Langerman, Carlos Seara, and Inmaculada Ventura. Facility location problems in the plane based on reverse nearest neighbor queries. *European Journal of Operational Research*, 202(1):99–106, 2010.

[18] Nick Roussopoulos, Stephen Kelley, and Frédéic Vincent. Nearest neighbor queries. In *SIGMOD*, pages 71–79, 1995.

[19] King Lum Cheung and Ada Wai-Chee Fu. Enhanced nearest neighbour search on the r-tree. *SIGMOD Record*, 27(3):16–21, 1998.

[20] Gísli R. Hjaltason and Hanan Samet. Distance browsing in spatial databases. *ACM Trans. Database Syst.*, 24(2):265–318, 1999.

[21] Vassilis Athitsos, Michalis Potamias, Panagiotis Papapetrou, and George Kollios. Nearest neighbor retrieval using distance-based hashing. In *ICDE*, pages 327–336, 2008.

[22] Yufei Tao, Ke Yi, Cheng Sheng, and Panos Kalnis. Quality and efficiency in high dimensional nearest neighbor search. In *SIGMOD Conference*, pages 563–576, 2009.

[23] Hanxiong Chen, Jianquan Liu, Kazutaka Furuse, Jeffrey Xu Yu, and Nobuo Ohbo. Indexing the function: An efficient algorithm for multi-dimensional search with expensive distance functions. In *ADMA*, pages 67–78, 2009.

[24] Hanxiong Chen, Jianquan Liu, Kazutaka Furuse, Jeffrey Xu Yu, and Nobuo Ohbo. Indexing expensive

functions for efficient multi-dimensional similarity search. *Knowl. Inf. Syst.*, 27(2):165–192, 2011.

[25] Flip Korn and S. Muthukrishnan. Influence sets based on reverse nearest neighbor queries. In *SIGMOD*, pages 201–212, 2000.

[26] Congjun Yang and King-Ip Lin. An index structure for efficient reverse nearest neighbor queries. In *ICDE*, pages 485–492, 2001.

[27] Yufei Tao, Dimitris Papadias, and Xiang Lian. Reverse knn search in arbitrary dimensionality. In *VLDB*, pages 744–755, 2004.

[28] Elke Achtert, Christian Böhm, Peer Kröger, Peter Kunath, Alexey Pryakhin, and Matthias Renz. Efficient reverse k-nearest neighbor search in arbitrary metric spaces. In *SIGMOD*, pages 515–526, 2006.

[29] S. S. Skiena. *The Algorithm Design Manual*, chapter Convex Hull, pages 351–354. Springer-Verlag, 1997.