集約k近傍の効率的な検索方法

塚原　拓人† 　董　 于洋† 　陳　 漢雄† 　古瀬　一隆†

† 筑波大学　〒 305–8571 茨城県つくば市天王台 1-1-1

E-mail: †s1520667@u.tsukuba.ac.jp, ††tou@dblab.is.tsukuba.ac.jp, †††chx@cc.tsukuba.ac.jp, ††††furuse@cs.tsukuba.ac.jp

あらまし　集約近傍（aggregate nearest neighbor, ANN）検索とは，クエリ集合 Q に対して，定められた距離関数に基づき，データベース P から Q との集約距離が最も近いオブジェクトを探す問題である.ANN はデータマイニングや情報検索，中でも特に地理情報システムにおいて多く利用されている．ANN 検索から k 近傍を求める AkNN への拡張は自然な要求である．今までには道路網上や平面上の L_1 距離などの限定されたケースでの AkNN についての研究が確認されたが，多次元に対応できず，また，より一般的なユークリッド距離への拡張は不可能であった．本研究では，汎用的な AkNN 検索の効率的なアルゴリズムを提案し，これを検証する．

キーワード　ANN, AkNN, データ索引, データ検索

1. 概　　要

近年，データマイニングや情報検索などに利用されている近傍探索は，情報検索の中でも重要な課題として活発に研究されている．近傍探索とは，データセットの中からクエリと最も類似度の高いオブジェクトを探索する問題である．近傍探索の一種である集約最近傍探索 (Aggregate Nearest Neighbor, ANN) とは，データセット P とクエリ集合 Q が与えられた時，データセットの中から Q との集 約距離が最近となるオブジェクトを探す問題である．

例えば，異なる場所にいる人たちがある会議に参加するとして，主催者が開催地を決めるとする． このとき，参加者たちのいる場所によって，集合場所までの距離や，そこに辿り着くまで の所要時間，かかる費用などがそれぞれ異なる． 主催者は，参加者の負担 (集合場所までの金額，時間等) が，できるだけ小さくなるように集合場所を決めることになる． もし，主催者が参加者の合計負担金額を重視するなら，すべての参加者の場所と集合地点との距離を計算し，合計距離の最も小さい場所を選ぶ．このような検索の解を求めるためには，既存の ANN 手法は計算時間が多くかかるという問題点がある． 特に，人数や集合候補地の数が増えると，計算量が大きく増えていく．

このように，ANN はデータをベクトルで表現し，それらの距離で類似度を測る． しかし，ANN では数値化されたパラメータや，その重みがユーザの意図を正確に反映しない場合がある． すると ANN で求められる解が，ユーザが求める解と一致しない可能性がある．

先程の例で言うと，集合場所までの距離の合計が厳密に最短となるような場所を探すことは可能であるが，ユーザーはある程度距離が短ければ，厳密に最近となるような場所よりも，集合場所の立地や，そこに辿り着くまでの交通手段の違い等の方が重要と考えるかもしれない．そのためには，最近となるオブジェクトただ 1 つを検索するのではなく，集約距離が短い集合場所候補を複数検索するといった方法が有効であると考えられる．

このような，ANN 検索から k 近傍を求める AkNN への拡張は自然な要求である．今までには道路網上や平面上の L_1 距離などの限定されたケースでの AkNN についての研究が確認されたが，多次元に対応できず，また，より一般的なユークリッド距離への拡張は不可能であった．そこで， 本研究では一般的な AkNN 検索に着目する．

既存の ANN 手法を直接 AkNN に適用することは不可能ではないが，このためには，この ANN 手法で求められた解を除いたデータに対して同じ ANN 手法を適用するという処理を $k-1$ 回繰り返す必要があり，非現実的である．

既存の ANN 手法を AkNN 手法に拡張した場合，ANN 手法は一般に現在の最良解候補との集約距離を用いて枝刈りを行うため，最低でも k 個のエントリ（かデータ点）を常時確保して初めて刈り取りが行える．これに対して本研究では，索引ノードがデータ点を k 個以上含んでいる場合に，クエリ集合 Q と，このノードとの距離を上界に用いた索引ノードの枝狩りすることを可能とした．これにより不要エントリの早期発見が可能になり検索の効率が向上する．本論文では，この性質を利用した汎用的な AkNN 検索の効率的なアルゴリズムを提案し，これを検証する．

2. 問 題 定 義

D 次元の空間にある両点 $p_1 = (x_1, \cdots, x_D)$ と $p_2 = (y_1, \cdots, y_D)$ の距離 $|p_1p_2|$ はマンハッタン距離やユークリッド距離などがある．本論文ではユークリッド距離を使う．p_1 と p_2 の距離 $|p_1p_2|$ は式 1 で定義する．

$$|p_1p_2| = \sqrt{\sum_{i=1}^{D} |x_i - y_i|^2} \tag{1}$$

そして，sum や max, min 等の単調増加集約関数 f を用いて，データ点 p とクエリ集合 $Q = \{q_1, q_2, \cdots, q_n\}$ との集約距

離 $adist\,(p,Q)$ を式 2 で定義する.

$$adist\,(p,Q) = f\,(|pq_1|\,,|pq_2|\,,\cdots,|pq_n|) = \underset{1\leqq i\leqq n}{f}\,(|pq_i|) \quad (2)$$

定義 1 (集約 k 近傍検索: AkNN 検索) データ集合 $P = \{p_1, p_2, \cdots, p_N\}$ とクエリ集合 $Q = \{q_1, q_2, \cdots, q_n\}$ と正の整数 k が与えられたとき, P の中から Q との集約距離が短い点を順に k 個検索することを集約 k 近傍 *(AkNN)* 検索という.

3. 関 連 研 究

集約最近傍 (aggregate nearest neighbor, ANN) 検索問題は, [1] にて集約関数 sum について定義され, [2] にて集約関数が max や min などの単調増加関数に拡張された. 前章で述べたように, 集約関数が $f = sum$ の場合, ANN の解はクエリ集合 Q との合計距離が最も短い点である. $f = max$ の場合, 各クエリ点との最長距離が最小となる点である. また $f = min$ の場合, 各クエリ点との最短距離が最小となる点である. [1] と [2] では, MQM(multiple query method) 手法,SPM(single point method) 手法, MBM(minimal bounding method) 手法の三つの ANN 手法が提案された. この三つの手法では, データ集合 P は R-tree と呼ばれる木構造によってインデックスされる.

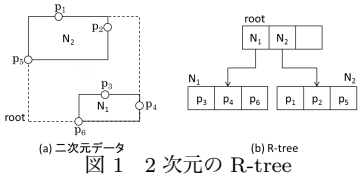


図 1　2 次元の R-tree

R-tree は B-tree のような木構造のデータ構造であり, 多次元情報の空間インデックスに使われる. 図 1 に 2 次元の R-tree の例を示す. 図のように, R-tree は入れ子状になった最小包囲矩形 (MBR) にデータ点を階層的に格納する. R-tree は低次元のデータにおいては高速に近傍を求めることができるが, データが高次元になると次元の呪いと呼ばれる現象が生じ, 近傍検索の性能は急激に下がり, 最悪の場合は計算量が $O(N)$ となる.

以下では距離が小さい順の最良優先探索を用いた MBM 手法について詳しく述べる.

3.1 MBM 手法

MBM(minimal bounding method) 手法は SPM 手法の問題点を解決するために提案された手法である. 基本的な考え方は SPM 手法と同様に, 枝刈りのルールを用いて, ANN の解を含んでいないエントリ（ノードか点）を刈り取り, クエリ集合 Q の ANN の解を高速に求める. MBM 手法では集約中心 q_m の代わりに, クエリ集合 Q と Q の最小包囲矩形 M が使われる.

MBM 手法は以下に示す 3 つの枝刈りルールを用いる.

ルール 1: ノード N_j と Q の最小包囲矩形 M の集約距離 $amindist(N_j, M)$ が $best\_dist$ より大きいか等しい場合, N_j の中には $best\_NN$ より良い解は含まれなく, 刈り取ってもよい. $f = sum$ の場合, $amindist(N_j, M)$ は N_j と M の最小距離 $mindist(N_j, M)$ の $n(= |Q|)$ 倍となる. $f = max$ か $f = min$ の場合, $amindist(N_j, M) = mindist(N_j, M)$ となる (式 3).

$$amindist(N_j, M) \geqq best\_dist \Leftrightarrow \begin{cases} n \cdot mindist(N_j, M) \geqq best\_dist & (f = sum) \\ mindist(N_j, M) \geqq best\_dist & (f = max \text{ or } min) \end{cases} \quad (3)$$

ルール 2: 点 p_j と M の集約距離 $amindist(p_j, M)$ が $best\_dist$ より大きいか等しい場合, p_j は ANN の解にならことはなく, 従って刈り取ってもよい. $f = sum$ の場合, $amindist(p_j, M)$ は p_j と M の最小距離 $mindist(p_j, M)$ の $n(= |Q|)$ 倍となる. $f = max$ か $f = min$ の場合, $amindist(p_j, M) = mindist(p_j, M)$ とする (式 4).

$$amindist(p_j, M) \geqq best\_dist \Leftrightarrow \begin{cases} n \cdot mindist(N_j, M) \geqq best\_dist & (f = sum) \\ mindist(p_j, M) \geqq best\_dist & (f = max \text{ or } min) \end{cases} \quad (4)$$

ルール 3: ノード N_j について, M との集約距離 $amindist(N_j, M)$ が $best\_dist$ より小さいが, Q との集約距離 $amindist(N_j, Q)$ が $best\_dist$ より大きいか等しい場合, N_j も刈り取ってもよい. $amindist(N_j, Q)$ はすべてのクエリ点 $q_i \in Q$ と N_j の最小距離 $mindist(N_j, q_i)$ の集約距離である. すなわち, $amindist(N_j, Q) = \underset{1\leqq i\leqq n}{f}\,(mindist(N_j, q_i))$. (式 5).

$$amindist(N_j, Q) \geqq best\_dist \Leftrightarrow \underset{1\leqq i\leqq n}{f}\,(mindist(N_j, q_i)) \geqq best\_dist \quad (5)$$

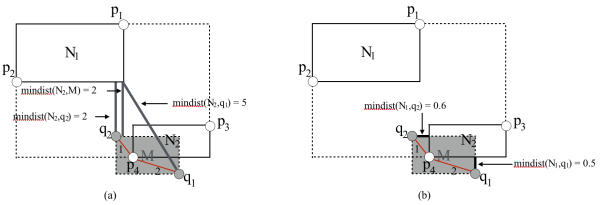


図 2　MBM の例 (集約関数 $f = sum$)

[1] と [2] にて, 著者たちは MQM 手法,SPM 手法,MBM 手法を提案し, MBM 手法はその中で最も高速な手法であることを示した. MBM 手法は今までで最速の手法である.

MBM 手法は Q の分布範囲が広いケースでも, ルール 3 が有効であるため SPM 手法より高速である. しかし, Q の範囲が広くかつ数が多い場合, すべてのクエリ点と R-tree にあるエントリからの距離を計算する必要があって, より時間が長くかかってしまう問題が残っている. また, 次元数が増えると, R-tree のノード同士が重なってしまい, 式 5 でも多くのノードを刈り取ることができない可能性がある.

3.2 その他の手法

ANN 検索に関する研究は他にも [5], [6], [9], [7], [8], [10], [11] 等が挙げられる. [5] では索引を使わない vp-ANN 手法と Projection-Based Pruning 手法を提案した. [6] では誤差を許

容する代わりに高速な近似 ANN 検索手法を提案した．[7] では楕円を用いて，[9] ではクエリ Q の重心との距離を用いてデータ集合 P を刈り取る手法を提案した．[8], [10] は道路網上にあるデータ集合の ANN 検索問題である．

3.3 AkNN 検索

AkNN 検索に関する研究は [12], [13] 等が挙げられが，それぞれ [12] では集約距離 $f = max$ における平面上の L_1 距離における AkNN 検索，[13] では道路網にあるデータ集合 P における AkNN 検索と，前述のようなユークリッド距離の，多次元にも対応するような汎用的な AkNN 検索に関する手法は提案されていない．

4. 提 案 手 法

前節で述べた MQM 手法,SPM 手法,MBM 手法は ANN 検索のための手法であり，AkNN 検索にそまま適用はできない．全く手を加えないまま AkNN 検索を実行しようとすると，単純に ANN 検索を k 回実行することになるが，解が求まるまでに時間を要する ANN 検索に対して，この方法はあまりに非現実的である．単純な拡張方法として，$best\_dist$ と $best\_NN$ の代わりに，それぞれを k 個を持つリストを保持し，常に距離順にソートする．MBM 手法における $best\_dist$ を用いた比較の箇所を距離リストの中で最も悪いものと比較し，解を更新するたびにリストをソートすることで，AkNN 検索に拡張することができる．本稿ではこの AkNN に拡張した MBM 手法を拡張 MBM 手法と呼ぶ．

MBM は Q の分布範囲が広いケースでも，ルール 3 により多くのノードを刈り取ることができるため高速である．しかし，ルール 3 ではルール 2 を満たす全てのノードに対して各クエリ点との距離を計算しなければならないため，R-tree のノードが重なる場合，特に高次元においては性能が低下する．これは AkNN 検索に拡張した場合においても同様である．また，最低でも最良解候補のデータ点を k 個確保するまでは刈り取りが行えない．

以下では，データ点まで到達していなくてもエントリを刈り取ることのできる，Q の分布範囲が広い場合や多次元における場合にも高速な AkNN 検索手法を提案する．

5. ノードとの最長距離による刈り取り

前述の通り，既存の ANN 手法を AkNN に拡張した場合，最低でも k 個のデータ点を確保しないと刈り取りが行えない．そこで，現在の最良解との距離 $best\_dist$ だけでなく，現在の最短距離を持つノードとの最長距離 $best\_maxdist$ を用いた刈り取りを導入する．

ルール 4: ノード N_{best} に含まれるデータ点 p の数が k 以下である場合，クエリ Q とノード N_j との集約距離 $amindist(Q, N_j)$ が $best\_maxdist$ より大きいか等しい場合，N_j の中には $best\_maxdist$ を持つノードに含まれるどの点よりも良い解は含まれず，刈り取って良い．

$$amindist(N_j, Q) \geqq best\_maxdist \Leftrightarrow \begin{cases} n \cdot mindist(N_j, Q) \geqq best\_maxdist & (f = sum) \\ mindist(N_j, Q) \geqq best\_maxdist & (f = max \text{ or } min) \end{cases} \quad (6)$$

これは，より早い段階から刈り取りが行えるため，高速であると期待できる．

6. クエリの重心との距離による刈り取り

MBM は Q の分布範囲が広くなると，クエリ Q の MBR である M と多くのノードや点が重なってしまう．その場合ルール 1 やルール 2 によって刈り取れるエントリが無くなり効率が落ちる．そこで Q の分布範囲に依らない刈り取りルールを導入する．

補題 6.1 ある点集合 Q とその重心 G に対して，データ点 p_j と G の集約距離は，p_j と Q との集約距離よりも小さいか等しい．

証明 6.1 $adist(p_j, G) = \sum_{i=0}^{n} \overrightarrow{Gq_i} + \sum_{i=0}^{n} \overrightarrow{q_ip_j} = \sum_{i=0}^{n} \overrightarrow{q_ip_j}$ *(G は Q の重心)*
$adist(p_j, G) = |\sum_{i=0}^{n} \overrightarrow{q_ip_j}| \leqq \sum_{i=0}^{n} |\overrightarrow{q_ip_j}| = adist(p_j, Q)$ ■

これを用いて，[9] と同様に，クエリ Q の重心 G を利用した，新たな刈り取りルールを導入する．

ルール 1':クエリの重心 G とノード N_j との集約距離 $amindist(G, N_j)$ が $best\_dist$ より大きいか等しい場合，N_j の中には $best\_NN$ より良い解は含まれず，刈り取って良い．

$$amindist(N_j, G) \geqq best\_dist \Leftrightarrow \begin{cases} n \cdot mindist(N_j, G) \geqq best\_dist & (f = sum) \\ mindist(N_j, G) \geqq best\_dist & (f = max \text{ or } min) \end{cases} \quad (7)$$

ルール 2':クエリの重心 G とデータ点 p_j との集約距離 $amindist(G, p_j)$ が $best\_dist$ より大きいか等しい場合，p_j は $best\_NN$ より良い解ではなく，刈り取って良い．

$$amindist(p_j, G) \geqq best\_dist \Leftrightarrow \begin{cases} n \cdot mindist(p_j, G) \geqq best\_dist & (f = sum) \\ mindist(p_j, G) \geqq best\_dist & (f = max \text{ or } min) \end{cases} \quad (8)$$

また，このルールは前項のルール 4 に対しても同様に適用できる．

補題 6.2 ある点集合 Q とその重心 G に対して，ノード N_j と Q の集約距離は，G と N_j との集約距離と Q と G の集約距離の和よりも小さいか等しい．

証明 6.2 三角不等式より $dist(q_i, N_j) \leqq dist(q_i, G) + dist(G, N_j)$
よって $amindist(Q, N_j) \leqq amindist(Q, G) + adist(G, N_j)$ ■

6.2 より，$best_maxdist = amaxdist(G, N_j) + adist(Q, G)$ としても刈り取り式は成り立つ．

そして，ルール 4 では，$best\_maxdist$ を更新するたびにノードと Q の集約距離を計算しなくてはならなかったが，補題 6.2 を用いることによって，以下のように，ノードとの集約距離を計算せずに刈り取りが行える．

ルール 4': 現在の最長距離が最良であるノード N_{best} に含まれるデータ点 p の数が k 以上である場合，クエリの重心 G とノード N_j との集約距離 $amindist(G, N_j)$ が $best\_maxdist$ より大きいか等しい場合，N_j の中には N_{best} に含まれる点よりも良い AkNN 解は含まれず，刈り取って良い．

$$amindist(N_j, G) \geqq best\_maxdist \Leftrightarrow \begin{cases} n \cdot mindist(N_j, G) \geqq best\_maxdist \ \ (f = sum) \\ mindist(N_j, G) \geqq best\_maxdist \ \ (f = max \text{ or } min) \end{cases} \quad (9)$$

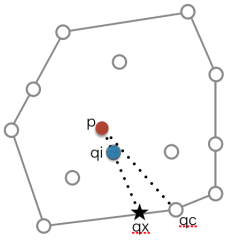
これらは，クエリ Q の分布範囲が広い場合でも，$amindist$ が 0 になることは少なく，より多くのエントリを刈り取ることができると期待できる．

7. クエリの前処理による手法

7.1 凸包の性質利用による手法 ($f = max$)

補題 7.1 (凸包定理) ある点集合 Q とその凸包に対して，Q の中で任意の点 p との距離が最大となる点 q_f は凸包頂点に含まれる．

証明 7.1 凸包頂点集合 Q_c と凸包頂点でない点 q_i があったとき，任意の点 p と q_i を直線で結ぶと，その直線と凸包が交差する点 q_x が存在する．q_i は p と q_x を結ぶ直線間に存在することになるため $|pq_i| \leqq |pq_x|$ である．また q_x の存在する辺の両端の凸包頂点 q_{c1}, q_{c2} に対して $|pq_x| \leqq |pq_{c1}|, |pq_x| \leqq |pq_{c2}|$ が成り立つ．よって $|pq_i| \leqq |pq_x| \leqq |pq_c|$ ■



$$|pq_i| \leq |pq_x| \leq |pq_c|$$

図 3 凸包定理

補題 7.2 ANN_{max} において，クエリ集合 Q の凸包頂点集合を Qc とすると，$ANN(Q) = ANN(Q_c)$ である．

証明 7.2 補題 *7.1* より，クエリ Q の凸包頂点集合 Q_c は，データ点 p に対して最大の距離をとるクエリ点 q_f を含む．ANN_{max} の定義より，$adist(Q, p) = adist(Q_c, p)$ であるため，$ANN(Q) = ANN(Q_c)$ である．■

従って，ANN_{max} においては，凸包を構成する点集合 Q_c に対してのみ距離計算を行えばそれ以外のクエリ点との距離計算が省略できるため，高速化が期待できる．

しかし，高次元における凸包の作成コストは高い．そのため凸包を作成せずに $Q_c \subseteqq Q_x$ となるような，なるべく小さい点集合 Q_x を取り出す．

まず，クエリ Q の MBR を作成する，MBR の辺上に存在する点は確実に凸包頂点に含まれるので Q_x に追加する．この点のうち，同じ頂点に隣接する 2 点を直線で結んだ時，その線より外側に他のクエリ点が存在するならば，そのクエリ点は凸包頂点候補となる．つまり，その 2 点を結んだ直線のよりも内側の点は凸包頂点にはなりえないということである．

そこで，Q_x に追加したクエリ点のうち，同じ頂点に隣接する 2 点で MBR を作成する．それらに囲まれた点は凸包頂点になりえないので除外する．Q_x に追加されたクエリ点と，除外されたクエリ点以外について同様の手順を行い，全てのクエリ点が Q_x に追加されるか除外されるまで続けることで Q_x を作成する．

この手法は，クエリの分布範囲が広いほど多くの点が除外できるため，効果的であると期待できる．

7.2 クエリのクラスタリングによる手法 ($f = sum$)

クエリを複数に分割することによって，ルール 1,2 を以下のように変形する．ただし，$Q_i \subseteqq Q$ は div 個に分割したクエリであり，G_i はそれらの重心，n_i は分割したクエリにそれぞれ含まれるクエリ点の数である．

ルール 1": ノード N_j と Q の重心 G_i の集約距離の和 $\sum_{i=0}^{div} amindist(G_i, N_j)$ が $best\_dist$ よりも大きいか等しい場合，N_j の中には $best\_NN$ よりも良い解は含まれず，刈り取ってもよい．

$$\sum_{i=0}^{div} amindist(N_j, G_i) \geqq best\_dist \Leftrightarrow \sum_{i=0}^{div} n_i \cdot mindist(N_j, G_i) \geqq best\_dist \ \ (f = sum) \quad (10)$$

ルール 2": データ点 p_j と Q の重心 G の集約距離 $amindist(G, p_j)$ が $best\_dist$ よりも大きいか等しい場合，p_j は $best\_NN$ よりも良い解ではなく，刈り取ってもよい．

$$\sum_{i=0}^{div} amindist(p_j, G_i) \geqq best\_dist \Leftrightarrow \sum_{i=0}^{div} n_i \cdot mindist(p_j, G_i) \geqq best\_dist \ \ (f = sum) \quad (11)$$

これらは，クエリを分割しない場合よりも，より ANN 解らしいエントリに対して良い値を取ると期待できる．特に，クエリの分布範囲が広い場合において，多くのエントリを刈り取ることができると期待できる．これらの提案手法の $f = sum$ における擬似コードを Algorithm1 に示す．

これらの提案手法の計算量は，クエリに前処理を施す手法は $f = sum$ と $f = max$ がそれぞれ $O(div * n)$，$O(n)$．データセットから k 近傍を求める計算量が両者ともに $O(n * N)$ で

ある．($f = max$ では n の代わりに，それよりサイズの小さい nx を用いる) よって $f = sum$,$f = max$ それぞれの計算量は $O(div * n + n * N)$，$O(n + nx * N)$ となる．それに対して，MBM の計算量は $O(n * N)$ である．しかし，提案手法は $f = sum$ においては一般に $div << N$ であり，$f = max$ においては n のサイズが減少するため，データセットから k 近傍を求めるための計算量と比べて，前処理のコストは非常に小さい．そのため，MBM との大きな差は無いと思われる．

Algorithm 1 提案手法 (f=sum) ($root$: R-tree root, Q: query set)

```
1: /* Q_i は Q の部分集合である */
2: /* G_i は Q_i の重心である */
3: /* H は amindist(N_j, Q) を昇順に並べた優先待ち行列である */
4: Node ← root; best_maxdist ← ∞; best_dist[k] ← ∞; best_NN[k] ← null
5: while Node ≠ null && amindist(Node, Q) < best_dist[k − 1] do
6:   if Node.isIndex then
7:     /* Node は R-tree の中間ノード */
8:     for ∀N_j ∈ Node do
9:       if amindist(N_j, G_i) < best_dist[k − 1] then
10:        if amaxdist(N_j, G_i) < best_maxdist then
11:          if amindist(N_j, Qc) < best_dist[k − 1] then
12:            H.push(N_j)
13:            best_maxdist ← amaxdist(N_j, G_i])+adist(G_i, Q)
               /* best_maxdist の更新 */
14:          end if
15:        end if
16:      end if
17:    end for
18:  else if Node.isLeaf then
19:    /* Node は R-tree の葉ノード */
20:    for ∀p_j ∈ Node do
21:      if amindist(p_j, G_i) < best_dist[k − 1] then
22:        if adist(p_j, Qc) < best_dist[k − 1] then
23:          best_NN[k − 1] ← p_j; best_dist[k − 1] ← adist(p_j, Q)  /* 候補 ANN の解の更新 */
24:        end if
25:      end if
26:    end for
27:  end if
28:  Node ← H.pop()
29: end while
30: return (best_NN)
```

8. 実　　験

8.1 実験環境

提案手法の有用性を示すために, 提案手法と MBM 手法を AkNN 検索に拡張した拡張 MBM 手法について比較実験を行った. 実験環境を以下に示す. また,R-tree の実装は spatialindex-src-1.8.1(注1) を利用した.

- OS: OS X Yosemite 10.10.5
- CPU: 2.6 GHz Intel Core i5
- メモリ: 8 GiB 1600 MHz DDR3
- 開発言語: C++

8.2 実験データ

実験は実データと合成データを用いて行った. 使用したデータは, 北アメリカの人口集落 (注2) の経緯度と, (0,1) の範囲にランダムで生成した一様分布データである. 実データは式 13 を用いて座標値を変換し,(0,1) の範囲に正規化した. ここで x_{max} と x_{min} はすべてのポイントの X 座標の最大値と最小値である. 同様に y_{max} と y_{min} は Y 座標の最大値と最小値である.

$$x^{'} = (x - x_{min})/(x_{max} - x_{min}) \quad (12)$$

$$y^{'} = (y - y_{min})/(y_{max} - y_{min}) \quad (13)$$

実験は問い合わせ集合 Q の数 n，クエリの分布範囲 $A(Q)$, 次元 D，k，$f = sum$ の場合はクエリ分割数 div を変化させ, 各手法の結果を比較する. 実験で用いた各パラメータの値は表 1 に示す.

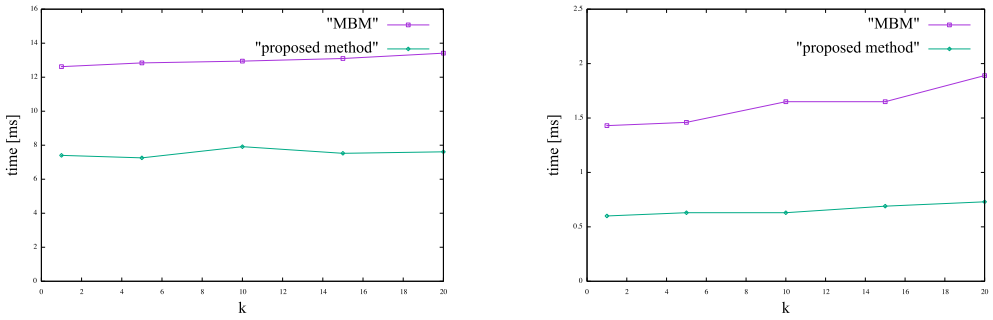
表 1　パラメータ

パラメータ	値
$n = \|Q\|$	16, 32, 64, 128, 256, 512
$A(Q)$	0.01, 0.09, 0.16, 0.25, 0.49, 1
D	2, 4, 8, 16
k	1, 5, 10, 15, 20
div	1, 5, 10, 15, 20, 25

8.3 実験結果

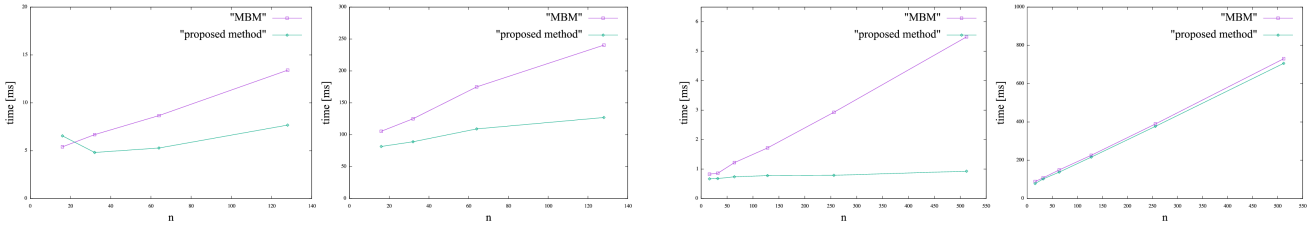
以下の実験結果は, ランダムに生成したクエリ点に対して 100 回の実験を行い, その平均を取ったものである.

(注1)：http://download.osgeo.org/libspatialindex/spatialindex-src-1.8.1.tar.gz
(注2)：populated places,http://www.rtreeportal.org/



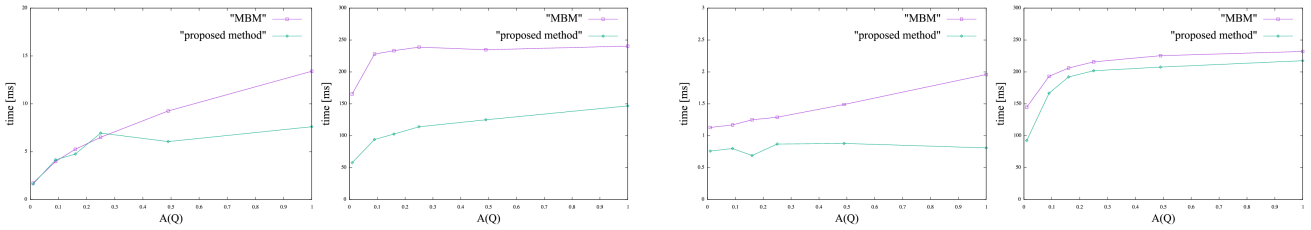
[a]$f = sum$ (div=20)　　[b]$f = max$

図 4　vary k $(n = 128, A(Q) = 1.0, dim = 2)$



[a]$f = sum$ (div=2)　　[b]$f = max$

図 6　vary n $(A(Q) = 1.0, dim = (2 or 8), k = 20, div = 20)$



[a]$f = sum$ (div=2)　　[b]$f = max$

図 7　vary $area$ $(n = 128, dim = (2 or 8), k = 20, div = 20)$

8.3.1　合成データ

図 4 は k を変化させたときの，拡張 MBM 手法と提案手法の実行速度の比較である．$f = max$ において MBM が k が増加するにつれて性能低下しているが，提案手法は変わらず高速であることが見て取れる．

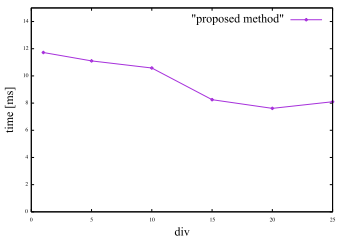
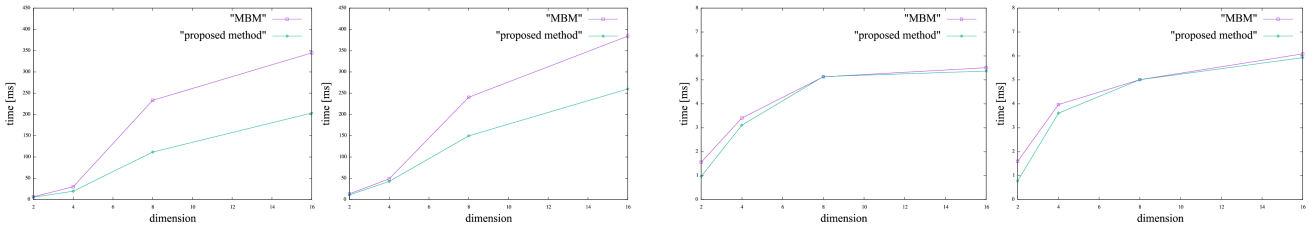


図 5　vary div $(n = 128, A(Q) = 1.0, dim = 2, k = 20)$

図 5 はクエリの分割数 div を変化させたときの提案手法の実行速度である．$div = 20$ までは速度は向上しているが，$div = 25$ では $div = 20$ よりも遅くなっている．これは，クエリの分割によって削減できる距離回数計算の数と，分割することで増えた各重心との距離計算回数が逆転したものと思われる．

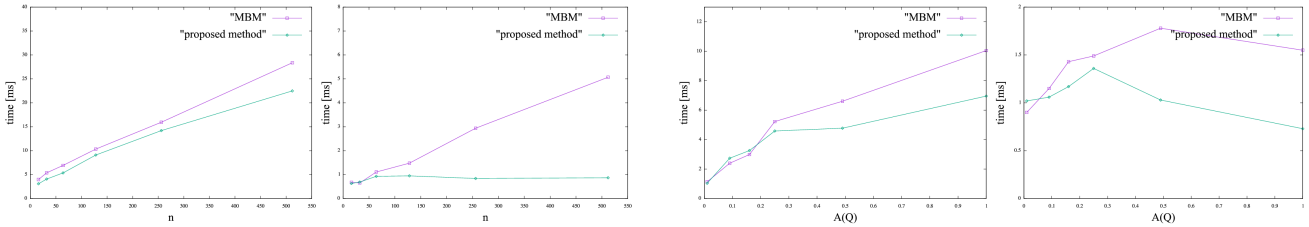
図 6 は $dim = 2$ と $dim = 8$ のとき，クエリ集合 Q のサイズ n を変化させたときの，拡張 MBM 手法と提案手法の実行速度の比較である．この結果から，sum は，n のサイズが大きくなるにつれて，拡張 MBM は性能が低下しているが，提案手法は変わらず高速であることが分かる．また，sum 右図の高次元においても同様に提案手法が高速であることが分かる．また，max は，左図 $dim = 2$ の場合，n のサイズが大きくなるにつれて，拡張 MBM と比べて提案手法が高速であることが分かる．しかし，右図 $dim = 8$ においては，提案手法はあまり効果がない事が分かる．これは，高次元においては，ほぼ全てのクエリ点が凸包頂点候補となってしまうためであると考えられる．

図 7 は，$dim = 2$ と $dim = 8$ のとき，クエリの分布範囲 $A(Q)$ を変化させたときの実行結果の比較である．sum は，左図 $dim = 2$ のとき，$A(Q)$ が小さい場合において，提案手法が拡張 MBM よりも遅くなる場合が存在する．これは，Q の分布範囲が狭い場合において，クエリを分割する手法があまり有効でないからだと考えられる．右図 $dim = 8$ のときにおいては，重心との距離を用いた刈り取りが有効であり，どの分布範囲においても高速である．max は，左図 $dim = 2$ のときは，$A(Q)$ が大きいほど提案手法が有効であることが分かる．これは，分布範囲が広いほうが凸包頂点候補となる点が少なくなるためだと考えられる．右図 $dim = 8$ においては，ほぼ全てのクエリ点



[a]$f = sum$ (div=2)　　[b]$f = max$

図 8　vary dim ($n = 128, A(Q) = (0.25 or 1.0), k = 20$)



[a]vary n (A(Q)=1.0, dim=2, k=20, div=20)　　[b]vary $area$ (n=128, dim=2, k=20, div=20)

図 9　実 デ ー タ

が凸包頂点候補となってしまいうため，分布範囲が広くなっても差は見られないが，重心距離との刈り取りによって一定の高速化が成されているのが分かる．

図 8 は，$A(Q) = 0.25$ と $A(Q) = 1.0$ のとき，次元 dim を変化させたときの，実行速度の比較である．sum はどちらにおいても提案手法が高速であり，次元や分布範囲に依らず提案手法が有効であることが分かる．

しかし max は，高次元になるにつれ，提案手法が拡張 MBM に近づいているのが分かる．これらの結果から，重心距離との刈り取りは，集約距離 max においてはあまり効果的でないと考えられる．

8.3.2　実 デ ー タ

図 9 は，実データに対して，クエリの分布範囲 $A(Q)$ とクエリサイズ n を変化させたときの実行速度の比較である．n については，左図の sum が合成データと比べて提案手法の効率が落ちているのが分かる．これは，クエリを分割する手法が，特に一様分布のような広い分布のデータに対して効率的であるためと考えられる．$A(Q)$ においては，sum，max 共におおよそ合成データと同じ傾向が見られる．

9. ま と め

本研究では ANN 検索の拡張として，AkNN 検索を提案した．AkNN の解を高速に求めるために，重心との距離による刈り取り手法，R-tree のノードとの距離を用いた刈り取り手法，クエリの前処理による手法を提案した．実験から，集約距離 sum においては，低次元かつクエリの分布範囲が狭い場合以外では，提案手法が高速であることを示した．しかし，集約距離 max においては，低次元では凸包の性質利用による手法が効果的であることを示したが，高次元においては効果が薄いことも分かった．

更なる効率化や，各手法の効率が落ちる場合での効果的な手法の提案が今後の課題となる．

文　　献

[1] D. Papadias, Y. Tao, K. Mouratidis and C. Kit Hui. *Group Nearest Neighbor Queries*. In: ICDE, pp.301-312(2004)

[2] D. Papadias, Y. Tao, K. Mouratidis and C. Kit Hui. *Aggregate Nearest Neighbor Queries in Spatial Databases*. ACM Trans. Database Syst. 30(2), 529-576(2005)

[3] H. Chen, R. Shi, K. Furuse and N. Ohbo. *Finding RkNN Straightforwardly with Large secondary Storage*. In: INGS, pp.77-82(2008)

[4] Y. Luo, J. Liu, C. Lian and H. Chen. *Finding RkNN by Compressed Straightforward Index*. In: ISKE, pp.279-284(2008)

[5] Y. Luo, H. Chen, K. Furuse and N. Ohbo. *Efficient Methods in Finding Aggregate Nearest Neighbor by Projection-Based Filtering*. In: ICCSA, pp.821-833(2007)

[6] 連 燦紅. *On efficient approximate aggregate nearest neighbor search.*. 筑波大学コンピュータサイエンス専攻修士論文,(2012)

[7] H. Li, H. Lu, B. Huang and Z. Huang. *Two Ellipse-based Pruning Methods for Group Nearest Neighbor Queries*. In: GIS, pp192-1999(2005)

[8] M.Yiu, N. Mamoulis and D. Papadias. *Aggregate Nearest Neighbor Queries in Road Networks*. IEEEE Trans. Knowl. Data Eng. pp820-833 (2005)

[9] S. Namnandorj, H. Chen, K. Furuse and N. Ohbo. *Efficient Bounds in Finding Aggregate Nearest Neighbors*. In: DEXA, pp.693-700(2008)

[10] L. Zhu, Y. Jing, W. Sun and P. Liu. *Voronoi-Based Aggregate Nearest Neighbor Query Processing in Road Networks* In: GIS, pp518-521(2010)

[11] F. Li, B. Yao and P. Kumar. *Group Enclosing Queries*. In: TKDE, pp, 1526-1540 (2011)

[12] Haitao Wang. *Aggregate-MAX Top-k Nearest Neighbor Searching in the L1 Plane*. Int. J. Comput. Geometry Appl. 25(1): 57- (2015)

[13] Maytham Safar. *Group K -Nearest Neighbors queries in spatial network databases*. Journal of Geographical Systems 10(4): 407-416 (2008)

[14] Camila Ferreira Costa, Javam C. Machado, Mario A. Nascimento, Jos Antnio Fernandes de Macdo. *Aggregate k-nearest neighbors queries in time-dependent road networks*. MobiGIS 2015: 3-12