高次元データ集合の最近傍探索問題に対する Flexible Distance-based Hashing に基づく厳密解探索手法

大崎　優也† 　若林　真一†† 　上土井　陽子††

† 広島市立大学情報科学部
†† 広島市立大学大学院情報科学研究科
〒 731–3194 広島市安佐南区大塚東 3 丁目 4 番 1 号
E-mail: †osakiyuya@lcs.info.hiroshima-cu.ac.jp,††{wakaba,yoko}@hiroshima-cu.ac.jp

あらまし　本稿では高次元データ集合の最近傍探索問題に対する Flexible Distance-based Hashing（FDH）に基づく厳密解探索手法を提案する．FDH は高次元データ集合の最近傍探索問題に対する近似解探索の発見的手法として知られている．本稿では，FDH に基づく近似解探索にバックトラック探索を導入することで厳密解探索手法に拡張する．本稿で提案する厳密解探索手法をプログラムとして実現し，計算機実験により従来厳密解探索手法である k-d 木探索手法および全探索手法と比較し，提案手法の有効性を示す．

キーワード　最近傍探索，高次元データ，Flexible Distance-based Hashing，厳密解探索，k-d 木探索

1 はじめに

最近傍探索問題はデータ集合 P とクエリ q に対し，定義された距離尺度に関してクエリに最も近いデータ（最近傍）をデータ集合 P から見つける問題であり，画像検索，機械学習，パターン認識など，多くの応用を持つ [4], [6], [7], [9], [12]．最近傍探索問題には多くの種別があるが，最も基本的でかつ重要な場合は，距離尺度がユークリッド距離であり，データ空間の次元 d が $d \geq 20$ のように非常に大きい場合である．本稿ではこうした高次元データ集合に対する最近傍探索問題の厳密解探索について考察する．

最近傍探索問題に対する単純な解法は全探索である．与えられたクエリに対し，データ集合のすべてのデータとの距離を計算し，距離が最小となるデータを解として出力する．全探索は大規模データ集合に対しては計算時間が多大となるため，より効率のよい解法を求めて多くの研究が行われてきた [12]．一方，最近傍探索問題に対する別のアプローチとして，解の精度を犠牲にすることで探索時間の短縮を実現する近似最近傍探索手法がある．最近傍探索問題の応用の多くにおいては，一旦データ集合が与えられるとデータ集合の更新はなく，同一のデータ集合に対して多数のクエリが与えられ，各クエリについて最近傍を求めることが一般的である．このため，データ集合に対して前処理を行って特定のデータ構造に変換した上で，与えられたクエリに対して厳密解や近似解を求めることが一般的である．

最近傍探索問題に対する既存の探索手法の多くは，木構造のデータ構造に基づく探索手法とハッシングに基づく近似最近傍探索手法の 2 つに分類することができる [10]．最近傍探索手法で使用される木データ構造の典型的な例は k-d 木，Rtree，B+-tree，Cover tree である．木データ構造に基づく最近傍探索手法の多くは厳密解（最近傍）を出力することを保証するが，高次元大規模データ集合に対しては効率的ではなく，しばしば全探索より探索時間が必要となることが知られている．一方，ハッシングに基づく近似最近傍探索手法としては，Locality-Sensitive Hashing（LSH）やスペクトルハッシングなどの様々な種類のハッシング技法が，良好な精度の近似最近傍を効率的に見つけるために提案されている [2], [5]．

本稿では，大規模高次元データ集合に対するハッシング技法に基づく厳密解探索手法を提案する．提案探索手法で用いるハッシング技法は Flexible Distance-based Hashing（FDH）である [11]．FDH は距離空間における大規模データ集合の類似データ検索のために提案されたハッシング技法である．与えられたデータ集合に対し，データ空間を領域と呼ばれる部分空間に分割し，領域ごとにデータを管理する．クエリが与えられると，FDH は最近傍が含まれる可能性が高いと予想される領域をハッシングにより選択し，領域に含まれるデータの中で，クエリに最も近いデータを最終探索結果として出力する．FDH は効率的な近似最近傍解探索を実現するが，解候補探索領域として最近傍を有する領域が選択されない可能性が常にあるという欠点を有する．本稿では FDH に基づく最近傍探索にバックトラック探索を導入することで厳密解を求める手法に拡張する．さらに，計算機実験により提案厳密解探索手法の有効性を示す．

本稿は以下のように構成される．第 2 章では，最近傍探索問題と Flexible Distance-based Hashing（FDH）について説明する．第 3 章では，FDH に基づく厳密解探索手法を提案する．第 4 章では，本稿で提案した最近傍探索手法の実験的評価を示す．最後に，第 5 章で結論を述べる．

2 準　　備

2.1 最近傍探索問題

最近傍探索問題は以下のように定義される．n 個の要素（デー

タ）を持つデータ集合 $P = \{p_1, \ldots, p_n\}$ が与えられたとする．P の各要素は d 個の属性 $(x_1, \ldots, x_d)$ からなり，各属性 x_i は $[L_i, U_i]$ の範囲内の実数である．ここで，L_i と U_i は属性 x_i の下限と上限を表す．定義から，P の各要素は d 次元空間 R^d のデータ点と見なすことができる．

データ集合 P の最近傍探索は次のように定義される．R^d 内のクエリ q が与えられたとき，クエリ q からユークリッド距離で最も近い点を P 内から探索する．以下では，R^d 内の 2 つのデータ点 s と t 間のユークリッド距離を $dist(s,t)$ と表す．

2.2 Flexible Distance-based Hashing（FDH）

Flexible Distance-based Hashing（FDH）は，高次元データ空間における最近傍探索のために提案されたハッシング技法である [11]．データ集合 $P = \{p_1, \ldots, p_n\}$ が与えられたとすると，この方法は最初に P から A 個のデータ点の集合 $A_n = \{a_1, a_2, \ldots, a_A\}$ を選択する．このとき，任意の 2 個のデータ点 a_i と a_j，$i \neq j$，に対し，その距離ができるだけ大きくなるようにデータ点を選択する．各 a_i はアンカーと呼ばれる．A_n の各アンカー a_i に対して，r_i で表される “半径” を $|P_{near}(r_i)| \simeq |P_{far}(r_i)|$ となるように決定する．ただし，$P_{near}(r_i) = \{p \in P | dist(a_i, p) \leqq r_i\}$，かつ，$P_{far}(r_i) = \{p \in P | dist(a_i, p) > r_i\}$ である．

アンカー集合を適切に設定することは，FDH を用いた最近傍探索において精度の良い近似解を得るために重要である．この問題に対するヒューリスティックな解法として反復改善手法が知られている [8]．この手法では，まずアンカー集合の初期解がランダムに選択される．次に，$dist(p,q)$ がすべてのアンカーの対の中で最小になるように，1 対のアンカー p および q を選択する．次に，p，q とは異なるデータ点 r が P からランダムに選択され，r と他のデータ点との間の最小距離が $dist(p,q)$ より大きい場合，p をアンカー集合から削除し，r を次のアンカー集合の要素に加える．そうでなければ r はアンカー集合の候補から外される．この手順を改善が見られなくなるまで繰り返し行う．

d 次元データ空間 R^d のデータ集合 P に対し，アンカー集合 A_n が与えられたとする．R^d 内のデータ点 p が与えられると，$BM(p) = b_1, b_2, \ldots, b_A$ で表される長さ A のビット列は以下のように定義される．

$$BM(p)[i] = b_i = \begin{cases} 0 & if \;\; dist(a_i, p) \leqq r_i \\ 1 & otherwise \end{cases} \qquad (1)$$

このビット列 $BM(p)$ をデータ点 p のビットマップと呼ぶ．ビットマップを使用して，d 次元データ空間 R^d を 2^A 個の部分空間に分割する．すなわち，$BM(p) = BM(q)$ が成り立つ場合，2 つの異なるデータ点 p と q が同じ部分空間に含まれる．以下では，各部分空間を領域と呼ぶ．各領域は対応するビットマップで識別する．例えば，$d = 2$, $A = 3$ の場合，2 次元データ空間は 8 つの領域に分割され，その境界は 3 つの円（$d = 3$ の場合は球，$d > 3$ の場合は超球）で表される．

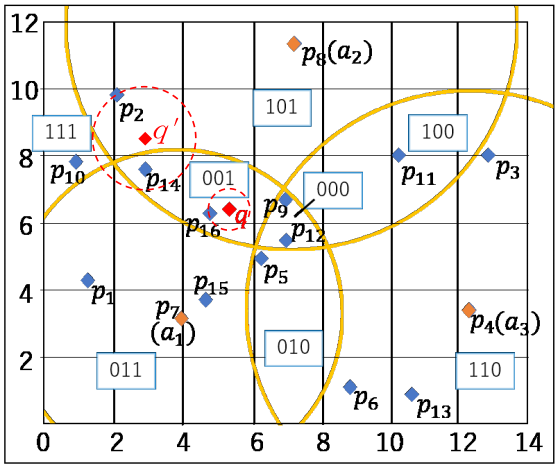

図 1　FDH の一例

d 次元データ空間のデータ集合 P に対して，P の各データが対応する領域ごとに管理されている場合，$BM(q)$ を計算することによって特定のクエリ q の最近傍データの近似解を効率的に探索できる．この探索方法は [11] で提案されたもので，Flexible Distance-based Hashing（FDH）と呼ばれる．例えば図 1 では，p_1，p_2，…，p_{16} の 16 個のデータ点があり，$p_7 = a_1$，$p_8 = a_2$，および $p_4 = a_3$ がアンカーである．クエリ q が与えられると，そのビットマップは 001 である（$BM(q) = 001$）．ビットマップ 001 のある領域には，p_{14}，p_{16} が含まれる．q とそれらの 2 つのデータ点との間のユークリッド距離を計算し，q に最も近い点として p_{16} を見つける．

2 つのデータ点がデータ空間内の近隣に存在する場合，2 つのデータ点が同じ領域に属する（すなわち，それらが同じビットマップを有する）可能性が高いので，FDH は局所的に敏感なハッシングと見なすことができる．FDH の利点は，データ空間の次元数に依存する FDH のパラメータがないため，FDH を用いた最近傍探索は任意の次元数に容易に適用できることである．しかしながら，クエリを含む領域が与えられたクエリに最も近いデータ点を含む保証はないので，厳密解を常に見つける保証はなく，この探索手法は最近傍探索問題に対する発見的手法である．

3　提案アルゴリズム

3.1　FDH を用いた近似最近傍探索の問題点

前節で説明したように，FDH を用いた近似最近傍探索手法は厳密解を見つける保証はないため，最近傍探索問題に対する発見的手法である．図 1 で示す例をもう一度考える．この図では，3 つのアンカーを含む 16 個のデータ点がある．クエリとして q' が与えられたとする．q' が存在する領域はビットマップ 101 である．クエリが存在する領域をクエリ領域と呼ぶ．クエリ領域には 2 つのデータ点があり，q' に最も近いデータ点として p_2 が最近傍の近似解として選択されるが，厳密解は領域

001 に属する p_{14} である．

そこで，データ空間のより広い範囲を探索するために，クエリ領域のビットマップとのハミング距離が H 以下のビットマップの領域をクエリ領域の隣接領域として探索するように FDH による近似最近傍探索を拡張する．例えば，ハミング距離 H を 1 として指定している場合，隣接領域のビットマップは，001，111，100 となり，厳密解 p_{14} を見つけることができる．ただし，この場合，クエリ領域を含む探索領域数は 4 であり，探索時間が増大し，探索が必要でない領域も探索する可能性がある．このように，クエリの最近傍がユークリッド距離的にはクエリの近くに存在しているのにも関わらず，H を大きくしない限り，FDH では見つけられない場合があるのが FDH に基づく近似最近傍探索の欠点である．著者らはこの欠点を解決するために，FDH に最遠 δ 近傍の概念を導入した改良探索手法を提案し，計算時間の増大を抑えつつ，より精度の高い近似解を見つけることを可能にしたが，常に最近傍を得られる保証はない [8]．

3.2 厳密解探索の基本的アイデア

FDH に基づく最近傍問題に対する厳密解探索手法の基本的アイデアを，図 1 に示す 2 次元空間における解探索を例に説明する．今，001 領域に属するクエリ q が与えられたとする．001 領域にはデータ点 p_{14}，p_{16} があり，q に最も近いデータ点 p_{16} が最近傍の近似解として出力される．q を中心とし，半径を q，p_{16} 間の距離とする円で囲まれる領域は 001 領域内になり，この円内には他のデータ点はないので，p_{16} よりさらにクエリ q に近いデータ点は存在せず，p_{16} は最近傍の厳密解であることがわかる．

次に，101 領域に含まれるクエリ q' が与えられたとする．101 領域にはデータ点 p_2，p_8 があり，クエリ q' に最も近いデータ点 p_2 が最近傍の近似解として出力される．しかしながら，q' を中心とし，半径を q'，p_2 間の距離 $dist(q', p_2)$ とする円で囲まれる領域は 101 領域だけでなく，001 領域，011 領域，111 領域と一部が重なっており，重なっている領域のデータ点が存在する可能性があるため，101 領域内のデータ点だけを探索対象とした場合は厳密解である保証がない．実際，この例では 001 領域に含まれるデータ点 p_{14} が厳密解となる．

上記の例から，FDH を用いた厳密解探索においては，与えられたクエリが存在する領域に含まれるデータ点を探索して，クエリとの距離が最も近い点を暫定解とし，クエリと暫定解の距離を半径とする円（高次元空間では超球）に含まれる他の領域のデータ点を逐次探索し，暫定解と超球の半径を更新しながら探索を続けることで，厳密解を求めることが可能であることが分かる．

3.3 FDH 探索木

前節で説明した FDH に基づく最近傍の厳密解を求める探索を効率よく実現するため，本稿では厳密解探索のためのデータ構造である FDH 探索木（FDH search tree）を新たに導入する．A をアンカー数とし，アンカー集合を $A_n = \{a_1, a_2, \ldots, a_A\}$ するとき，A 個のアンカーに対する FDH 探索木は高さ $A+1$ の完全 2 分木である．図 2 にアンカー数が 3 の場合の FDH 探索木を示す．木の根節点がアンカー a_1 に対応し，根節点以外の木の内節点は根節点からの距離に対応してアンカー a_2，a_3，…，a_A に対応する．アンカー a_i に対応する木の内節点をレベル i の節点と呼ぶ．根節点はレベル 0 であり，葉節点はレベル $A+1$ である．

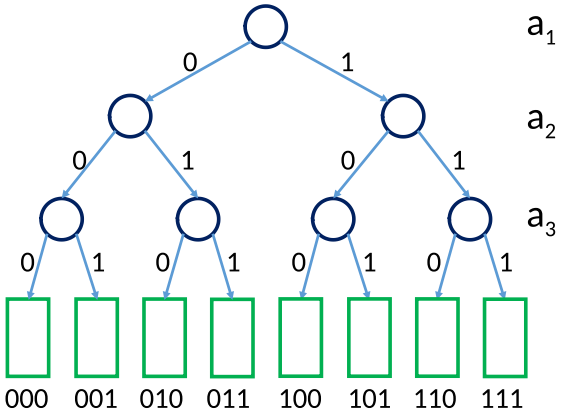

図 2　FDH 探索木

木のレベル i の内節点はレベル $i+1$ の節点に対して 0 枝と 1 枝と呼ぶ枝で接続されている．0 枝はアンカー a_i を中心とし半径 r_i で定義される円（超球）の内側の領域に対応し，1 枝は外側の領域に対応する．この対応付けにより，葉節点はアンカー集合で定義される 2^A 個の領域に対応する．図 2 の例では，8 個の葉節点は 000 領域から 111 領域に対応する．

FDH 探索木の各葉節点はその葉節点に対応する領域に属するデータ集合を記憶しているものとする．図 2 では，例えば 000 領域の葉節点は 000 領域に含まれるデータ集合が記憶されている．図 1 で表される FDH の場合，データ点 p_9 と p_{12} が記憶されている．

3.4 FDH 探索木を用いた厳密解探索の概要

FDH 探索木を用いた厳密解探索の概要を示す．厳密解探索は FDH 探索木を用いたバックトラック探索により実現される．クエリ q が与えられると最初に q が含まれる領域を求め，領域に対応する FDH 探索木の葉節点に記憶されているデータ集合の中で，q と最も距離の近いデータを暫定解 x とする．また，q と暫定解との距離を求めて q_x とする．q を中心とし半径を q_x とする円（超球）を暫定解領域と呼ぶ．クエリ q を含む領域にデータが存在しない場合は q_x を無限大（∞）に設定する．

暫定解が求まると，FDH 探索木のレベルを 1 だけ小さくし（バックトラック探索，木においては親節点に戻る操作に対応），その節点の 0 枝あるいは 1 枝を辿る必要があるかどうかを判定する．辿る必要がある場合はその枝を辿り，必要がなければさらにレベルを小さくする．0 枝あるいは 1 枝をたどって葉節点に到達した場合は葉節点に含まれるデータ集合を対象として暫定解の更新が必要かを確認し，更新する必要がある場合は暫定解を更新する．この操作を繰り返して，最終的に根節点まで戻ってきて，さらに辿る枝がなくなればアルゴリズムを終了し，

そのときの暫定解を厳密解として出力する．

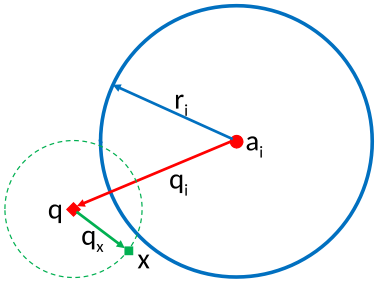

図 3　0 枝の探索条件

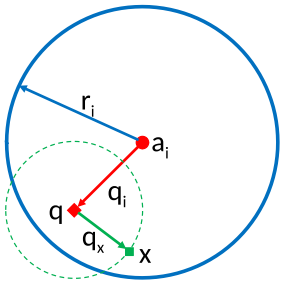

図 4　1 枝の探索条件

3.5　探索条件

前節において FDH 探索木をバックトラック探索をする際に判定する 0 枝と 1 枝の探索条件を以下に示す．まず，レベル i の探索木の内節点において 0 枝を探索する条件を示す．レベル i の内節点はアンカー a_i に対応している．クエリ q が a_i を中心として半径 r_i の円（超球）の外に存在していると仮定する．このとき，0 枝に対応する円内の探索が必要になるのは図 3 に示す場合となる．すなわち，アンカー a_i からクエリ q までの距離を q_i とし，q と暫定解 x との距離を q_x とするとき，q を中心とし，半径 q_x の円（超球）が a_i を中心とし，半径 r_i の円（超球）と重なる領域を持つ場合である．式で表すと以下のようになる．

$$r_i \geqq q_i - q_x \tag{2}$$

クエリ q が a_i を中心として半径 r_i の円（超球）の内側にある場合は式 (2) が無条件になりたつ．よって，0 枝を探索する条件は式 (2) となる．

次に，レベル i の探索木の内節点において 1 枝を探索する条件を示す．クエリ q が a_i を中心として半径 r_i の円（超球）の内側に存在していると仮定する．このとき，1 枝に対応する円外の探索が必要になるのは図 4 に示す場合となる．すなわち，a_i から q までの距離を q_i とし，q の暫定解との距離を q_x とするとき，q を中心とし，半径 q_x の円（超球）が a_i を中心とし，半径 r_i の円（超球）の外側と重なる領域を持つ場合である．式で表すと以下のようになる．

$$r_i < q_i + q_x \tag{3}$$

クエリ q が a_i を中心として半径 r_i の円（超球）の外側にある場合は式 (3) が無条件になりたつ．よって，1 枝を探索する条件は式 (3) となる．

なお，FDH 探索木のバックトラック探索において，1 度探索した枝は 2 度探索する必要はないため，探索した枝についてはフラグを立てて再度の探索をしないようにする．また，内節点を根節点とする FDH 探索木の部分木において，部分木のすべての葉節点がデータを含まない場合，その部分木では暫定解が更新されることはないので，その内節点を探索済みとして親節点にバックトラックする．

3.6　データ構造

最近傍探索問題に対する提案厳密解探索手法のデータ構造を示す．図 5 に FDH 探索木の各節点のデータ構造を示す．FDH 探索木はこの節点のデータ構造をポインタで相互に結合することで実現する．

図 6 に，入力データ集合 P と領域データのデータ構造を示す．探索対象となる n 個の d 次元データは $n \times d$ の 2 次元配列で記憶され，配列のインデックスがデータの識別番号であると仮定する．

入力データ集合はあらかじめ与えられたアンカー集合で決定された領域ごとにリスト構造で記憶される．領域データは領域番号でアクセスされる．領域データ構造から FDH 探索木の葉節点へのポインタも用意されている．このようなデータ構造とすることで，データ点の更新（追加，削除）にも容易に対応できる．

3.7　提案手法の詳細

図 7 に提案手法の詳細を示す．前提として，探索対象のデータ集合 P に対し，アンカー集合 A_n が決定され，アンカー集合に基づいて FDH 探索木が構築され，各領域に属するデータ集合も決定され，FDH 探索木の対応する葉節点にリンクされているものとする．

提案手法は与えられた FDH 探索木の節点に対し，その節点を根とする部分木の中で暫定解を探索する関数 find_nearest_neighbor が中心であり，前節で説明した分枝条件に従って 0 枝あるいは 1 枝を辿って，部分木を探索する．葉節点に到達した場合は暫定解の更新を試行する．クエリ q に対する最近傍の暫定解 x が更新されるごとに暫定解領域は小さくなる．

提案手法のメイン関数である nearest-neighbor はクエリ q が含まれる領域番号を求めた後，関数 find_nearest_neighbor を呼び出す．メイン関数に制御が戻って来た場合，親節点に戻ることが可能な場合は親節点にバックトラックし，探索を継続する．バックトラックができなくなったら提案手法は終了する．

3.8　提案手法の時間計算量

n 個のデータを含むデータ集合 P に対してアンカー集合 A_n

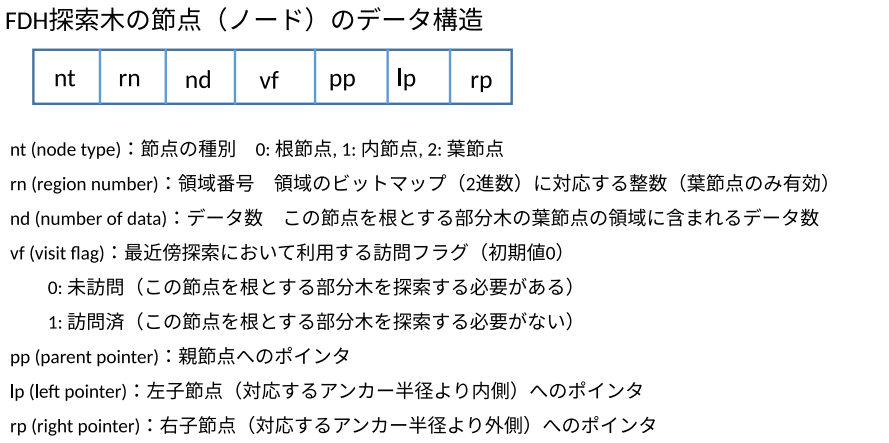

図 5　FDH 探索木のデータ構造

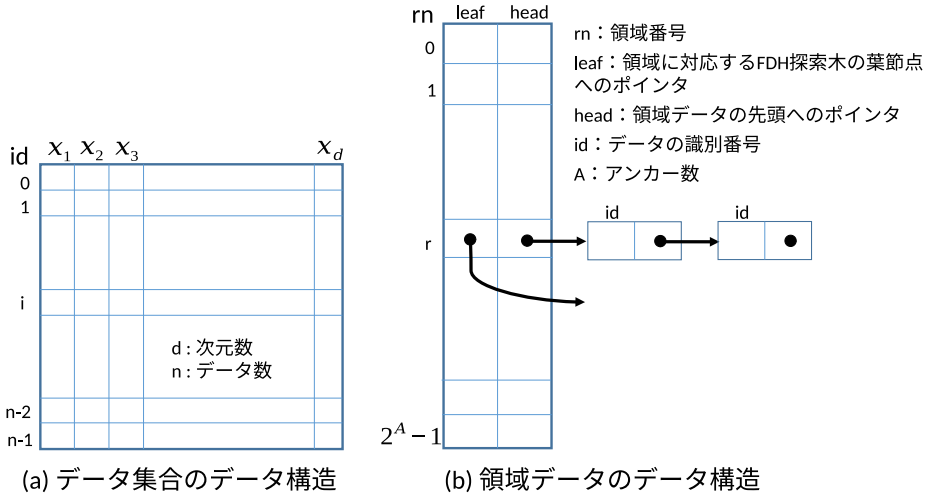

図 6　データ集合と領域データのデータ構造

を決定し，FDH 探索木と領域ごとのデータ部分集合を構成することは $O(n)$ 時間かかる．本稿では同じデータ集合に対して，多数のクエリによる最近傍探索を繰り返し実行することを想定しているため，FDH 探索木と領域データの作成に必要な時間計算量は提案手法の評価には含めない．

アンカー数を A とすると，領域数は 2^A となるので，領域ごとのデータ数は，均等にデータが領域に分散されていると仮定すると，各領域は $O(n/2^A)$ 個のデータを持つ．クエリ q に対して最短で最近傍が求まるのはバックトラック探索なしで求まる場合であり，その場合の提案手法の時間計算量は $O(n/2^A)$ となる．バックトラックによって探索される葉節点が定数個に収まる場合も同じ計算量となる．

一方，探索に要する計算時間が最悪となる場合はバックトラック探索が続いて，最終的にすべての葉節点を探索する場合であり，この場合，FDH 探索木のすべての節点も探索することになる．FDH 探索木の節点数は $2^{A+1}-1$ であるが，通常は $2^{A+1}-1 < n$ なので，この場合の提案手法の時間計算量は $O(n)$ となり，全探索と同じ時間計算量となる．ただし，提案手法は全探索と比較すると探索のオーバヘッドが大きいので，実際の計算時間は全探索と比較すると大幅に大きくなる．

4　実験的評価

この章では，本稿で提案した最近傍探索問題に対する厳密解探索手法（以下では EA-FDH と呼ぶ）の最近傍探索に要する計算時間を，従来厳密解探索手法である k-d 木 [3] と全探索を比較対象として実験的に評価し，考察を行う．

4.1　実 験 環 境

EA-FDH と k-d 木と全探索を C 言語を用いてプログラム実装し，CPU：AMD Ryzen 5 2600 3.40GHz，OS：Windows10 の PC を使用し，コンパイラ：gcc8.1.0，最適化オプション：-O2 でコンパイルして実行した．

4.2　実験結果と考察

実験に用いるデータ集合のデータは，各次元の値域を $(0, 100)$

入力：
　d 次元データの集合 $P = \{p_1, p_2, \ldots, p_n\}$, $p_i =< x_1^i, x_2^i, \ldots, x_d^i >$
　P に対して構成された FDH 探索木とアンカー集合 $An = \{a_1, a_2, \ldots, a_A\}$
　各アンカーに対する超球（円）の半径 r_1, r_2, ..., r_A
　クエリ $q =< q_1, q_2, \ldots, q_d >$
出力： クエリ q の最近傍データ $x \in P$

find_nearest_neighbor(p, i); /* ポインタ p が指す節点を根節点とする部分木の葉節点で最近傍を見つける関数 */
入力:
　p FDH 探索木の節点へのポインタ
　i 現在の節点レベル（アンカー番号）
　q クエリ（グローバル変数）
出力（グローバル変数）:
　q_x クエリ q と現在の暫定解 $x \in P$ の距離（アルゴリズム終了時には q と最近傍の距離）
　x 暫定解（アルゴリズム終了時には q の最近傍）
begin
　if $i = A + 1$ then /* p が指す FDH 探索木の節点が葉節点の場合 */
　　$rn \leftarrow$ 葉節点に対応する領域番号;
　　if 領域 rn に含まれるデータ数 > 0 then
　　　$t \leftarrow$ find_nn(rn); /* 領域番号 rn の領域中の q の最近傍 t */
　　　if dist$(q, t) < q_x$ then $x \leftarrow t$; $q_x \leftarrow$ dist(q, t); /* 暫定解 x の更新 */
　else /* p が指す FDH 探索木の節点が内節点の場合 */
　　if (FDH($*$FDH($*p$).lp).vf= 0) and (FDH($*$FDH($*p$).lp).nd > 0) then /* 左子節点（半径 r_i の内側）が未訪問の場合 */
　　　if dist$(q, a_i) - q_x \leqq r_i$ then /* a_i は i 番目のアンカー */
　　　　find_nearest_neighbor(FDH($*p$).lp,$i + 1$); /* 左子節点を探索（再帰呼び出し） */
　　if (FDH($*$FDH($*p$).rp).vf= 0) and (FDH($*$FDH($*p$).rp).nd > 0) then /* 右子節点（半径 r_i の外側）が未訪問の場合 */
　　　if dist$(q,a_i) + q_x > r_i$ then /* a_i は i 番目のアンカー */
　　　　find_nearest_neighbor(FDH($*p$).rp,$i + 1$); /* 右子節点を探索（再帰呼び出し） */
　FDH($*p$).vf $\leftarrow 1$; /* 節点の訪問フラグを 1 にする */
end;

アルゴリズム nearest-neighbor; /* 最近傍探索のメイン関数 */
Step 0： /* 初期化 */
　FDH 探索木のすべての節点の訪問フラグを 0 にする;
　$q_x \leftarrow \infty$; /* クエリ q からの探索半径（暫定解までの距離）を ∞ にする */
Step 1： /* クエリ q が含まれる領域を求める */
　$rn \leftarrow$ region_number(q); /* クエリ q が含まれる領域の領域番号を求める */
　$p \leftarrow$ 領域番号 rn に対応する FDH 探索木の葉節点へのポインタ;
　$i \leftarrow A + 1$; /* FDH 探索木の節点のレベル（対応するアンカー節点のインデックス） */
Step 2： /* p が指す FDH 探索木の節点を根とする部分木の最近傍を求める*/
　find_nearest_neighbor(p, i);
Step 3： /* バックトラック */
　if $i = 1$ then アルゴリズムの終了 /* x が最近傍 */
　else /* 親節点にバックトラック */
　　$i \leftarrow i - 1$; /* 節点レベルを -1 */
　　$p \leftarrow$FDH($*p$).pp; /* p に親節点へのポインタを設定 */
　　Step 2 に戻る;

図 7　提 案 手 法

の範囲内の実数とし，ランダムに生成した．データ集合に対するクエリの数は 10,000 とした．計算時間の評価においては，探索木の生成などの初期設定は含まず，クエリを与えてから厳密解が出力されるまでを計測対象としている．得られる解は必ず厳密解で精度は保証されているため，各探索手法の評価は探索に要する計算時間で行う．EA-FDH のアンカー数 A は予備実験により，$A = 13$ に設定して実験を行った．

4.2.1　ランダムクエリ

クエリを一様乱数を用いてランダムに生成し，3 つの解法の処理時間を比較する．図 8 はデータ数を 100,000，横軸は次元数，縦軸は処理時間 (秒) で 10,000 個のクエリに対する探索時間の合計を表す．

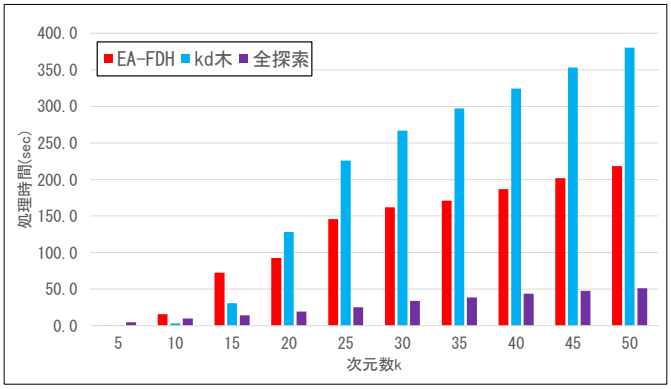

図 8　一様乱数によってクエリを生成した場合

図 8 から，次元数が一定以上大きくなると EA-FDH は k-d 木と比べて短時間で結果を出力することが分かる．しかし，EA-FDH と k-d 木の両方が全探索と比べて大幅に出力に時間がかかる結果となった．これは次元数が大きくなるにつれてデータ空間が極端に大きくなり，データ集合が非常に疎となり，与えられるクエリのほとんどが探索対象のデータ点と遠い位置に設定されるため，バックトラック探索が頻繁に起こり，データ集合の大部分を探索するために，探索のオーバーヘッドが増大することが原因と考えられる．

4.2.2　正規乱数を用いて生成したクエリ

最近傍探索の応用においては，データに近いクエリが与えられた場合には可能な限り短い探索時間で厳密解を出力することが望ましい場合がある．そこで，正規乱数を用いてデータに近いクエリをランダムに生成し，探索に要する計算時間を評価する．はじめにクエリ q の生成方法について説明する．データ集合 P からランダムにデータ p を選択し，p の各次元 x_i について，平均 x_i，標準偏差 σ の正規乱数を生成し，クエリ q の各次元の値とすることで，p に近いクエリ q を生成する．

図 9 はデータ数を 100,000，標準偏差 $\sigma = 1.0$，横軸は次元数，縦軸は処理時間 (秒) で 10,000 個のクエリに対する探索時間の合計を表す．

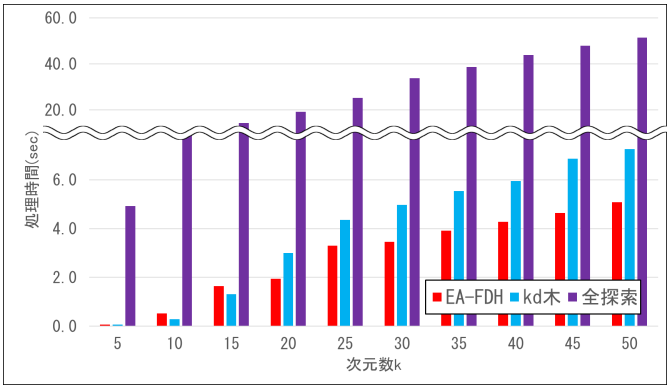

図 9　$\sigma = 1$ の正規乱数によってクエリを生成した場合

図 10 はデータ数を 100,000，標準偏差 $\sigma = 3.0$，横軸は次元数，縦軸は処理時間 (秒) で 10,000 個のクエリに対する探索時間の合計を表す．

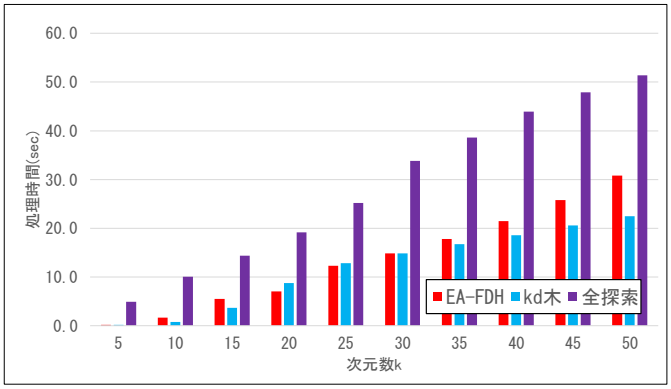

図 10　$\sigma = 3$ の正規乱数によってクエリを生成した場合

図 11 はデータ数を 100,000，標準偏差 $\sigma = 5.0$，横軸は次元数，縦軸は処理時間 (秒) で 10,000 個のクエリに対する探索時間の合計を表す．

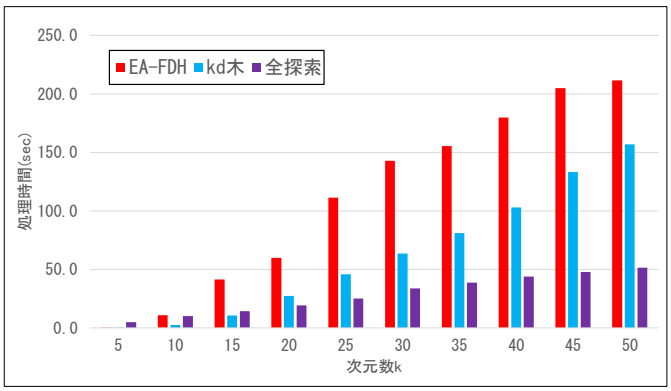

図 11　$\sigma = 5$ の正規乱数によってクエリを生成した場合

正規乱数を用いてクエリを生成した場合について考察する．正規乱数の標準偏差 σ が 1.0 の場合は図 9 より，EA-FDH が全探索や k-d 木と比べて短時間で結果を出力が分かる．標準偏差 σ が 3.0 の場合は図 10 より，EA-FDH は全探索と比べると EA-FDH は短時間で結果を出力できているが，k-d 木と比べると結果の出力にかかる時間が長くなっている．標準偏差 σ が 5.0 となると図 11 より，EA-FDH は全探索と k-d 木の両方の処理時間より長い結果となった．この結果から，与えられたクエリに近いデータ点が存在する場合は，EA-FDH は全探索や k-d 木と比べて短時間で結果を出力できるため，データに近いクエリが与えられる応用や，クエリに近いデータが存在する場合は短時間で結果が欲しい応用などで有効であると考えられる．

4.2.3　考　　察

前節の実験結果が示すように，バックトラック探索がほとんど生じない場合，提案アルゴリズムは全探索と比較して大幅な計算時間の短縮が可能である．このため，データ点に近いクエリが与えられる場合が多い応用においては，高速検索が期待できる．また，クエリに非常に類似したデータがデータ集合に存在する場合は短時間で探索結果を得たい応用においても有用である．

一方，クエリがランダムに与えられ，データ点から近くない

クエリも多い場合は計算時間の短縮は期待できない．特に，高次元データ集合に対する最近傍探索の場合，いわゆる「次元の呪縛」[12] の影響を受けて，全探索より計算時間が大幅に増えることが実験的に確認された．

探索木を利用した厳密解探索手法である k-d 木 [3] も，提案探索手法と同様の実験結果となった．k-d 木と提案手法を比較すると，k-d 木も 2 分探索木に基づく探索手法であることは提案手法と同じであるが，k-d 木は与えられたデータ集合に依存した木構造であることが提案アルゴリズムと異なる．提案アルゴリズムはアンカー集合によってデータ空間を領域に分割し，分割された領域ごとにデータを管理しているので，検索対象のデータの更新（追加，削除）が生じても柔軟に対処できるという特徴を持つ．一方，k-d 木は探索木のすべての節点が与えられたデータ集合のデータ点に対応しているので，検索対象のデータ点の更新が生じた場合，探索木の再構成が必要となる．このため，検索対象となるデータ集合の更新頻度が高い場合は提案アルゴリズムが有利である．

5 おわりに

本稿では，大規模高次元データ集合に対する最近傍探索のための厳密解探索手法を提案した．提案厳密解探索手法は距離ベースのハッシング手法である Flexible Distance-based Hashing (FDH) に基づいており，FDH に基づく近似最近傍探索手法にバックトラック探索を導入することで，厳密解法に拡張した．提案手法の計算時間を理論的に評価すると共に，その特徴を考察した．さらに，計算機実験により提案探索手法の有効性を示した．提案探索手法は探索対象のデータ集合のいずれかのデータ点に近いクエリが与えられることが多い応用や，与えられたクエリに近いデータ点が存在する場合にはできるだけ短時間で最近傍を出力することが望まれる応用において有用である．

今後の課題として，ランダムに生成したデータ集合だけでなく，最近傍探索のベンチマークデータ [1] や最近傍探索の実際の応用から得られたデータに対する実験的評価がある．アンカー集合の選択が探索性能に与える影響についても検討することも興味深い．

文　献

[1] ANN-benchmarks, http://ann-benchmarks.com/index.html

[2] Vassills Athitsos, Michalis Potamias, Panagiotis Papapetrou, George Kollios, “Nearest neighbor retrieval using distance-based hashing,” *Proc. IEEE International Conference on Data Engineering*, pp.327–336, 2008.

[3] Jon Louis Bentley, “Multidimensional binary search trees used for associative searching,” *Communications of the ACM*, Vol.18, No.9. pp.509–517, 1975.

[4] Gérard Biau, Luc Devroye, *Lectures on the Nearest Neighbor Method*, Springer, 2015.

[5] Junfeng He, Regunathan Radhakrishnan, Shih-Fu Chang, Claus Bauer, “Compact hashing with joint optimization of search accuracy and time,” *Proc. 2011 IEEE Conference on Computer Vision and Pattern Recognition*, pp.753–760, 2011.

[6] Alexander Hinneburg, Charu C. Aggarwal, Daniel A. Keim, “What is the nearest neighbor in high dimensional spaces?,” *Proc. 26th VLDB Conference*, pp.506–515, 2000.

[7] Yoonho Hwang, Bohyung Han, Hee-Kap Ahn, “A fast nearest neighbor search algorithm by nonlinear embedding,” *Proc. 2012 IEEE Conference on Computer Vision and Pattern Recognition*, pp.3053–3060, 2012.

[8] Yuri Itotani, Shin’ichi Wakabayashi, Shinobu Nagayaya, Masato Inagi, “An Approximate Nearest Neighbor Search Algorithm Using Distance-based Hashing,” Proc. 29th International Conference on Database and Expert Systems Applications (DEXA2018), 7B, 2018.

[9] Marius Muja, David G. Lowe, “Scalable nearest neighbor algorithms for high dimensional data,” *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol.36, No.11, pp.2227–2240, 2014.

[10] Peter N. Yianilos, “Data structures and algorithms for nearest neighbor search in general metric spaces,” *Proc. Fourth annual ACM-SIAM symposium on Discrete algorithms*, pp.311–321, 1993.

[11] Man Lung Yiu, Ira Assent, Christian S. Jensen, and Panos Kalnis, “Outsourced similarity search on metric data assets,” *IEEE Trans. Knowledge and Data Engineering*, Vol.24, No.2, pp.338–352, 2012.

[12] 和田俊和, “最近傍探索の理論とアルゴリズム”, 情報処理学会研究報告, Vol.2009-CVIM-169, No.13, 2009.