次元縮小を用いた拡張可能グリッドファイルによる高次元データの逆最近傍検索

三好　涼介† 　三浦　孝夫† 　塩谷　勇††

† 法政大学 工学部 電気電子工学科　〒 184–8584 東京都小金井市梶野町 3-7-2
†† 産能大学 経営情報学部　〒 259–1197 神奈川県伊勢原市上粕屋 1573
E-mail: †{i04r3245,miurat}@k.hosei.ac.jp, ††shioya@mi.sanno.ac.jp

あらまし　逆最近傍検索とは，質問点が最近傍点となるデータ集合中のすべての点を見つけるものであり，ドキュメントデータベース，地理データベース等の多数のアプリケーションで必要性が高まっている．本稿では，次元縮小を用いた拡張可能グリッドファイルにおいて，k-最近傍検索と範囲検索を実行することにより，逆最近傍検索の近似解答を効率良く得られることを示す．

キーワード　多次元データ処理システム，拡張可能グリッドファイル，次元縮小，逆最近傍検索

Querying Reverse Nearest Neighbor High Dimensionality Data on Extensible Grid Files using Dimensionality Redcution

Ryosuke MIYOSHI†, Takao MIURA†, and Isamu SHIOYA††

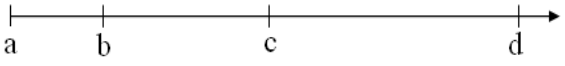
† Dept.of Elect.& Elect. Engr., HOSEI University　3-7-2, KajinoCho, Koganei, Tokyo, 184–8584 Japan
†† Department of Management and Information Science, SANNO University　1573, Kamikasuya, Isehara city, Kanagawa 259–1197 Japan
E-mail: †{i04r3245,miurat}@k.hosei.ac.jp, ††shioya@mi.sanno.ac.jp

Abstract　Given a query point q, *Reverse Nearest Neighbor Query* retrieves all points of the database to which q is the nearest neighbor. This operation is interested in a wide range of applications such as document database and geografic database. In this paper, We get approximate RNN data by examining k- Nearest Neighbor Query and Range Query on Extensible Grid Files using Dimensionality Reduction. And we show that our technique can answer RNNQ efficiently by some experimental results.

Key words　Multi-dimensional data processing system , Extensible Grid Files , Dimensionality Reduction , Reverse Nearest Neighborhood Query

1. 前　書　き

逆最近傍検索とは，質問点が最近傍点となるデータ集合中のすべての点を見つけるものである．例えば下図の各データは，a が b , b が a , c が b , d が c を最近傍点とするので，それぞれの逆最近傍点は a が b , b が a と c , c が d , d が無しとなる．



逆最近傍検索は最近傍検索の応用として解決できるが，低次元データでさえこの検索を実行するのに膨大な時間がかかる．最も単純な方法は，それぞれの点データの最近傍点をあらかじめ求めておき，その距離が質問点との距離より短くなるデータを算出する方法である．しかしこの計算量は $O(n^2)$ となり，有用な方法ではない．そのため，現在いくつかの手法が提案されている．

RNN 木 [12] は R 木の派生であり，各データ p_i の最近傍距離 $dnn(p_i)$ を事前に算出し，データを中心とする半径 dnn の球をデータオブジェクトとして R 木を生成する．与えられた質問点 q に対し，q と重なる球を持つデータすべてが逆最近傍検索の答えとなる．検索自体は完全一致検索と同様であるが，生成時において事前に最近傍距離を算出する点では線形走査と変わらない．

また，更新が起こる場合にも最近傍検索の処理が必要であるため，動的なデータベースでは実行可能ではない．そのため

RNN 木と，NN 木と呼ぶ最近傍検索のための R 木を組み合わせて使用することで動的データベースに適応させている．しかし，更新が起こるたびに RNN 木と NN 木の両方を修正しなければならないため，非常に効率の悪いものとなる．

Rdnn 木 [25] は，RNN 木のノードに部分木中のデータの最大の最近傍距離を付与することで RNN 木での最近傍検索を高速化し，更新処理の効率を向上させる構造である．しかし最近傍距離の前計算が必要という問題は引き継いでいる．

文献 [22] では，幾何学的なアプローチで逆最近傍検索を実行する．2 次元データでは逆最近傍点は 6 個以下であることを利用し，質問点を中心としてデータ平面を 6 つに等分割する．それぞれの分割平面で質問点の最近傍点を算出し，質問点の逆最近傍点かを決定する．この手法は 2 次元では効率よく動作するが，次元が高くなると分割空間数が大きく増加するため，高次元データでは実行不可能である．

高次元での逆最近傍検索のアルゴリズムとして文献 [1] が挙げられる．この手法では，最近傍点と逆最近傍点との間に相互関係があるという観測結果を元に，k-最近傍検索により逆最近傍点の候補集合を作成し，ブール範囲検索と呼ばれる範囲内のデータの有無を調べる検索を用いてフィルタリングを行う．この手法により得られる解答は完全ではないが，高い精度を保つことが実験により知られている．

本稿ではこのアルゴリズムを，次元縮小を用いた拡張可能グリッドファイルに適用させる．拡張可能グリッドファイル [14](Extendsible Grid File, 以下 EGF) は管理領域をデータ分布に従って変動させることで偏りを防ぐ多次元索引構造である．グリッドファイルに拡張可能ハッシュ法を適応することで，挿入および検索時のグリッド空間の分割に伴う参照回数の急激な上昇を回避している．また，バケットへの索引にデータ内在判定と MBR 情報を与えることで，空バケットの存在に起因する検索コストを低減している．さらにデータの次元縮小を用いることで，実データのような非一様に分布する高次元データに対して高い検索精度，検索効率を実現している．これにより，逆最近傍検索の近似解答を効率良く得られることを示す．

2 章では EGF 構造と処理の流れを述べ，3 章では高次元データの次元縮小について述べる．4 章で次元縮小を用いた EGF における検索の手法について論じる．5 章では実験結果を示し，6 章で結びとする．

2. 拡張可能グリッドファイルによる多次元データの検索

2.1 データ表現とデータ構造

EGF では，n 次元の多次元データに対し，各次元の値を 2 進数で表現しそれぞれの上位 P_n 桁を用いる．この数がバケットへの索引となり，グリッドファイルにおけるグリッドに相当する．さらに個々の索引に対して，その索引の示すバケットのデータ内在判定と，バケット内に 1 つだけ MBR の存在を許すときの MBR 情報を付与する．この配列をディレクトリと呼ぶ．対応するバケット，データ内在判定，MBR の頂点の座標を M_L,M_H(ただし $M_L = (l_1, l_2, ..., l_n)$,$M_H = (h_1, h_2, ..., h_n)$,$l_i \leqq h_i$) と表す．ディレクトリは $2^{\sum P_n}$ 個の索引からなる．

また，索引の桁数 P_n はバケット数に応じて伸縮する．そして，その切り出された値より索引が指定され，その索引に対応するバケットを取り出し各種操作を行う．挿入操作の場合はディレクトリが更新される可能性があり，またデータのないバケットにデータが挿入されるとき，もしくは MBR の領域外の座標のデータが挿入されるときも更新される．

空バケットにデータを挿入する場合，そのバケットを示す索引すべてのデータ内在判定を更新する．MBR の領域外の座標のデータを挿入する場合，当該バケットを示すすべての索引の MBR も更新する．

[例 1] 図 1 では，x 軸, y 軸共に値の上位 1 桁をディレクトリへの索引としている．索引 (0,0) はバケット A，(0,1) はバケット B，(1,0),(1,1) はバケット C へのポインタとなる．また，(0,0) はバケット A の，(0,1) はバケット B の，(1,0),(1,1) はバケット C のデータ内在判定とそれぞれのバケット内の MBR 情報を持つ．ここで $R = (51, 117) = (0110011, 1110101)$ となるデータ R の挿入を考える．

$P_x = P_y = 1$ であるから R_x, R_y の上位 1 桁を取り出し索引を得る．$(0, 1)$ の示すバケット B を取り出し挿入を行う．もしバケット B が空バケットである場合，$(0, 1)$ のデータ内在判定を真にし，MBR 情報を $((51, 117), (51, 117))$ に更新する．また，バケット B の MBR 情報が $((40, 120), (50, 125))$ である場合，$((40, 117), (51, 125))$ に更新する．

2.2 データ更新による構造変化

バケットにはあらかじめ最大容量 B が決められており，一定のデータ数を超えるとあふれが生じる．このときデータの偏りを防ぐため，バケットの管理領域を分割する．バケットが複数の索引から参照されているときはバケット自体を分割すればよいが，単一索引から参照されている場合にはディレクトリを拡張する必要がある．このため，EGF では管理領域の桁数 p_n を記憶する．

バケットにおいてデータ挿入によってあふれが生じる場合，すべての軸で $P = p$ となるバケットであればディレクトリ拡張操作が，それ以外のバケットならばバケット分割操作を行う．バケット分割時はあふれの生ずるバケットの管理する領域を分割軸で 2 等分割する．片方を前のバケットが，もう一方を新しく作成するバケットが管理するとし，挿入データも含めてデータを移動させる．ディレクトリ拡張時は拡張軸の索引を 2 倍に拡張してからバケット分割を行う．データ移動後，索引をそれぞれのバケットに合わせて更新する．バケット分割時の分割軸の選択は，データ分布の大きい軸を分割軸とする．ディレクトリ拡張時の拡張，分割軸の選択は P_n が最小の軸を選択する．

[例 2] 図 1 のバケット C があふれた場合バケット分割操作が行われ図 2 の状態となり，バケット B があふれた場合ディレクトリ拡張操作が行われ図 3 の状態となる．

削除操作は挿入とほぼ逆の手順で行うことができる．ただし，削除時におけるバケットの合併やディレクトリの縮小は即時に実行しなくてよい．例えば，影響の少ない時間まで延期するこ

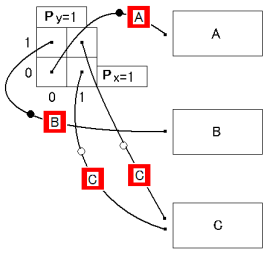


図 1　拡張可能グリッドファイル

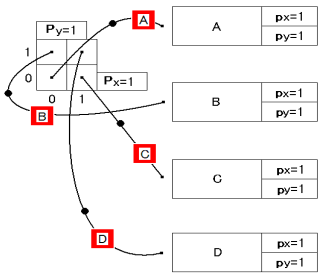


図 2　バケット C 分割後のキー分布

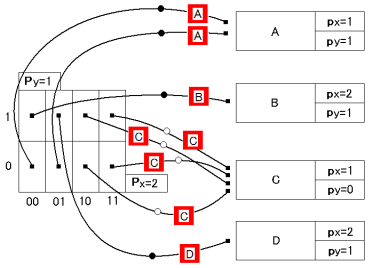


図 3　ディレクトリ拡張後のキー分布

とができる．

2.3　拡張可能グリッドファイル構造の検索手法

2.3.1　完全一致検索

完全一致検索とは，各次元のキーをすべて指定し，これをすべて満たす空間情報を求める検索をいう．検索点のそれぞれ上位 P_n 桁を取り出し索引を得る．取り出した索引において判定を行う．索引に対応するバケットの内在判定が真で，かつ検索点が MBR 内にあるなら，バケットにアクセスしバケット内のすべてのデータに対して比較を行う．

2.3.2　範 囲 検 索

範囲検索とは，ある指定した範囲に含まれるすべてのレコードを求める検索をいう．一般に検索範囲はすべての軸 i について，$LOW \leqq a_i \leqq HIGH$ の形で指定される．検索方形領域の対角の頂点の上位 P_n 桁を取り出し索引とし，グリッド空間に射影させた範囲にある索引について判定を行う．索引に対応するバケットの内在判定が真で，かつ検索範囲が MBR と重なるバケットをアクセスリストに入れ，それ以外のバケットにはアクセスしない．すべての索引の判定終了後，アクセスリストにあるバケットのデータに対して実際の判定を行う．

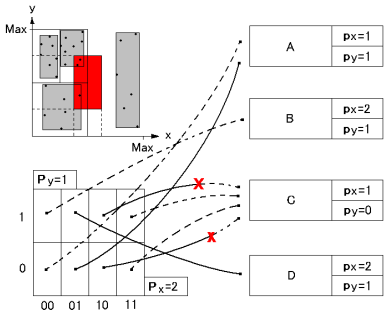


図 4　範 囲 検 索

2.3.3　k-最近傍検索

各次元のキーをすべて指定して，最も近い位置にあるレコードから k 番目に近い位置にあるレコードまでを求める操作を k-最近傍検索といい [20]，$k = 1$ であるものを特に最近傍検索と呼ぶ．本稿では文献 [3] で記述される距離優先アルゴリズムと同様の手法を用いる．

まず，質問点 Q とそれぞれのバケットの MBR との最小距離を算出し，最小距離の昇順に整列し，アクセスリストとして保存しておく．このとき，質問点 Q が MBR 内にある場合はそのバケットとの距離は 0 とし，バケット内にデータがない場合はそのバケットを除外する．

アクセスリスト作成後，これに従い MBR の近い順でバケットにアクセスし，バケット内のデータを候補集合リストに入れ，Q との距離の昇順で候補集合リストを整列する．候補集合数が k を満たしてからは k 番目の距離 L_k を基準とし，Q との距離が L_k より短いデータを k 番目のデータと入れ替え，再び候補集合リストを整列する．

次のバケットにアクセスする前に，この時点での L_k より長い最小距離をもつバケット内のデータは，すべて現在の k 番目の候補データより遠くにあるため，このバケットへのアクセスは無駄なものとなる．そのため，このようなバケットをアクセスリストから除外する．

枝刈り後にアクセスすべきバケットが残っている場合は同様の手順を繰り返し，アクセスすべきバケットが無くなった時点で終了し，候補集合リスト内のデータが Q の k-最近傍点となる．

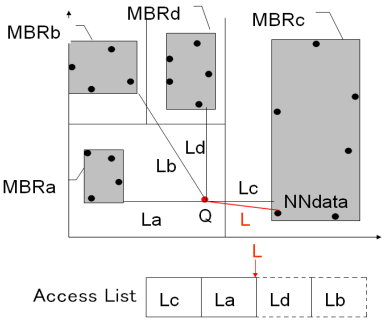


図 5　最近傍検索

3.　高次元データの次元縮小

前章では EGF の構造とその処理について述べた．これらの多次元索引構造は低次元データでは効率よく動作するが，ある次元を超えると空間全体を探索しなければならなくなるため，高次元データの処理は全件走査の処理に劣る性能となる [23]（”次元の呪い (Curse of Dimensionality)”と呼ばれる）．

本稿では，次元縮小を用いたデータを扱う索引構造を使用する．この手法は情報検索に酷似しており，これにより得られる解答は必ずしも正解ではないが，検索コストを低減することができる．本章ではこの次元縮小法について述べる．

3.1 潜在的意味索引付け

潜在的意味索引付け（Latent Semantic Indexing，以下 LSI）[13], [19] では，まず特異値分解（SVD）によって次元縮小のための射影行列を求める．次元数 d，データ数 N のデータ行列 X の SVD は，次の式で表される．

$$X_{d\times N} = U_{d\times r} S_{r\times r} V^T_{r\times N} \quad (1)$$

行列 U，V は直交行列で，それぞれの列ベクトルを左特異ベクトル，右特異ベクトルと呼ぶ．行列 S は対角行列であり，$S_{11} \geqq S_{22} \geqq \cdots$ という性質を持つ．これらの対角要素を特異値と呼ぶ．

次に，データ行列を j 次元 $(j << d)$ に縮小する．LSI における次元縮小は，次の計算で行う．

$$X^{SVD}_{j\times N} = U^T_j X \quad (2)$$

U_j は大きさ $(d \times j)$ の行列で，行列 U から最初の j 個の左特異ベクトルを抜き出したものであり，j 個の左特異ベクトルは，最も大きな j 個の特異値に対応している．同様に検索点も，同じ射影行列を用いて縮小次元空間に射影することができる．

ここで，データの次元縮小に伴い生じる近似誤差は特異値によって算出でき，X と X_j の違い $Diff$ は，以下の式で表される．

$$Diff = \frac{||X - X_j||_F}{||X||_F} = \frac{\sqrt{S^2_{j+1j+1} + \cdots + S^2_{rr}}}{\sqrt{S^2_{11} + S^2_{22} + \cdots + S^2_{rr}}} \quad (3)$$

LSI は検索精度を維持したまま次元を大きく縮小することができるため，検索効率と検索精度を両立することができる．

しかし，LSI ではデータの更新に対して特異値分解のための再計算が必要となる．このため，更新が頻繁に行われるデータを扱う場合には LSI は有用な手段ではない．この問題に対し，本稿ではデータの重心のみで特異値分解を行う Centroid SVD [9] を使用する．

3.2 Centroid SVD

LSI と Centroid SVD の計算過程は，すべてのデータ行列で特異値分解を行うか，重心のみで特異値分解を行うかの違いだけである．Centroid SVD [9] では，まずクラスタリング手法によりデータの重心を算出する．次に，重心に対して特異値分解を行い，行列 U,S,V を算出する．最後に，行列 U_j を使用してデータ行列 X を次元縮小する．データ更新を行う場合，算出した射影行列 U により更新データを次元縮小し，データ更新を実行する．

この手法を用いることにより，多少の更新を許容する射影行列が算出でき，かつ特異値分解に要する計算時間を短縮できる．

4. 次元縮小を用いた拡張可能グリッドファイルにおける検索手法

本章では，次元縮小を用いた拡張可能グリッドファイル [15] の検索手法を論じる．ここでは，n 次元のデータを j 次元に縮小するとして考える．構築および更新については，2.1 , 2.2 で述べた方法を縮小次元データに対して行えばよい．また次元縮小する前のデータは，縮小次元データに対応して保存しているものとする．

4.1 完全一致検索

まず検索点 Q を縮小次元空間に射影し，検索点 Q' を算出する．次に，検索点 Q' のそれぞれ上位 P_j 桁を取り出し索引を得る．取り出した索引において判定を行う．索引に対応するバケットの内在判定が真で，かつ検索点が MBR 内にあるなら，バケットにアクセスしバケット内のすべての縮小次元データに対して比較を行う．Q' と合致する縮小次元データがあれば，その元データと検索点 Q を比較し，真にデータの有無を判定する．

4.2 範囲検索

次元縮小を用いた EGF による範囲検索は，辺長 r の立方体領域の場合のみを考える．まず検索範囲の中心点 Q を縮小次元空間に射影し，縮小次元における検索範囲の中心点 Q' を算出する．次に，検索方形領域の対角の頂点の座標を算出する．このとき，縮小次元空間での検索範囲は次元縮小により生じる誤差を含む範囲を検索する．よって，この 2 頂点 LOW,$HIGH$ は式（3）の $Diff$ を用いて以下で表すことができる．

$$LOW_i = Q'_i - \frac{r}{2} \cdot (1 + Diff)(i = 1, \cdots, j)$$

$$HIGH_i = Q'_i + \frac{r}{2} \cdot (1 + Diff)(i = 1, \cdots, j)$$

この範囲を用いて縮小次元内での EGF で範囲検索を行い，$LOW_i \leqq D'_i \leqq HIGH_i (i = 1, \cdots, j)$ を満たす縮小次元データを算出する．最後にそれらの元データより，$Q_i - \frac{r}{2} \leqq D_i \leqq Q_i + \frac{r}{2} (i = 1, \cdots, n)$ を満たすデータを真の範囲内のデータとして算出する．

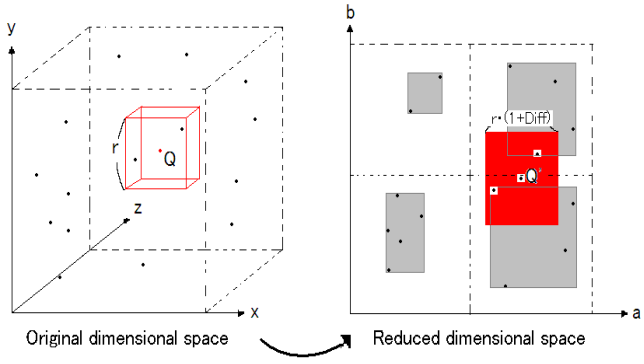


図 6　次元縮小後の範囲検索

4.3 ブール範囲検索

質問点 Q を中心とする半径 r の球の範囲内に，データがあるかどうかを求める検索をブール範囲検索という．範囲内に 1 つでもデータが見つかればただちに検索を終えることが出来る点が，範囲検索との大きな違いである．

まず検索範囲の中心点 Q を縮小次元空間に射影し，縮小次元における検索範囲を求めアクセスすべきバケットを決定する．このとき，ランダムにバケットにアクセスするよりも，範囲内のデータを保持する可能性の高いバケットからアクセスするほうが，I/O 回数が小さくなるのは明らかである．よって本稿では，Q' と MBR との最小距離が小さいバケットから順にアク

セスする．

バケットにアクセス後，得られるデータの元データより，データが元の次元での検索範囲内にあるかどうかを判別し，あればその時点で終了し真を返し，なければ同様の手順を繰り返す．すべてのバケットにアクセスし，元の次元での検索範囲内にデータがない場合は偽を返す．

4.4 k-最近傍検索

まず検索点 Q を縮小次元空間に射影し，縮小次元空間での検索点 Q' を得る．次に縮小次元内での EGF で k-最近傍検索を行うが，ここで次元縮小の誤差を考慮して，k'-最近傍検索 $(k' \leqq k)$ を実行する．得られる k' 個の候補集合データの元データより，それぞれのデータの Q との距離を計算し，距離の小さい上位 k 個のデータを k-最近傍検索の解答として算出する．

この次元縮小を用いた EGF における k-最近傍検索を，k/k'-最近傍検索と呼び，k' の値としてどの程度が良いかは，実験 1 によって調査する．

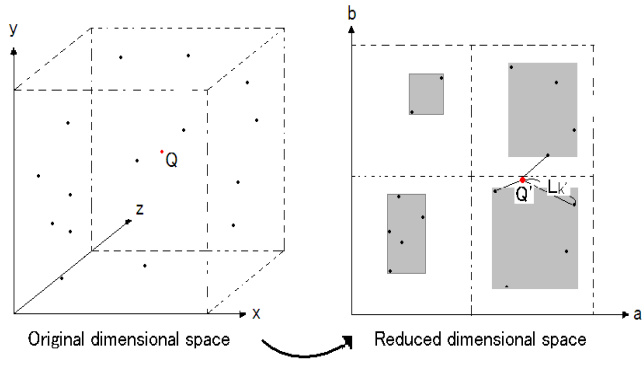


図 7　次元縮小後の最近傍検索

4.5 逆最近傍検索

逆最近傍検索は 3 つの段階からなる．

まず次元縮小を用いた EGF で質問点 Q の k-最近傍検索を行い，逆最近傍点の候補集合を作成する (k-最近傍検索処理)．次元が高くなるにつれて，逆最近傍点の理論的な数は大きく増加するが，実際には 10 を超えることは稀で，617 次元でさえ最大で 7 という結果が文献 [1] において観測されている．よって，k の値は 5 から 10 の範囲で十分であると考える．

次に候補集合内のデータのみで，候補集合内のそれぞれのデータを質問点として最近傍検索を実行する (フィルタリング処理 1)．もし質問点以外の最近傍点を持つデータがあれば，そのデータは Q の逆最近傍点ではないのでこれ以降の段階から除外できる．この作業は，データ集合数と比べて非常に小さい k 個のデータ集合での最近傍検索であるので，キャッシュメモリ内で十分に処理できる．そのため，I/O なしで候補集合をさらに小さくすることができる．

最後に，残った候補集合内のデータすべてに対しブール範囲検索を実行する (ブール範囲検索処理)．このときの縮小次元空間内での検索範囲は r により近似せずとも，候補データ P_i の次元縮小データ P_i' と Q' により，P_i を中心とし一辺の長さを $P_i' - Q'$ 間の距離とした正方形領域を算出できる．よって，この処理ではこの範囲を調査する．この処理で真に調査すべき元の次元空間での領域は，P_i を中心とした半径 $P_i - Q$ の円領域であるので，縮小次元空間での正方形領域内において Q との距離が $P_i - Q$ 間の距離より短い元データを持つデータがあるかを調査し，結果として偽を返すデータのみを Q の逆最近傍点とする．

5. 性 能 評 価

本章では，次元縮小を用いた EGF による逆最近傍検索の有用性を実験により評価する．

5.1 実 験 環 境

実験では，100 次元の非一様分布データ 120,000 点を使用する．これはニューヨーク，ボストン，フィラデルフィア周辺の郵便アドレス(注1)の分布を利用して生成する．これを 100000 と 20000 点に分割し，それぞれ初期生成と追加挿入として使用する．一括生成では，まず 100000 点から 100，239, 100000 の重心を算出し，特異値分解を用いて行列 U, S, V を求める．その後，行列 U より 4 次元, 6 次元, 8 次元に次元縮小し，各データに対して一括生成を行う．追加挿入では，残りの 20000 点を先に求めた行列 U を用いて 4 次元, 6 次元, 8 次元に次元縮小し，そのデータを先に構築したそれぞれの一括生成ファイルに投入する．

本実験ではディレクトリをメモリ上に配置し，I/O を無視できるとみなす．また，質問点 100 点は同様の分布に従い生成する．1 バケットを 1 ページとし，$B = 8$ Kbyte として実験を行う．

5.2 実　験　1

実験 1 では，次元縮小を用いた EGF での k/k'-最近傍検索を実行し，k' の変化による F 値とページアクセス回数について観測する．また，適度な F 値となる k' でのページアクセス回数について考察を行う．

データは重心数 100 の行列 U により次元縮小した 4 次元, 6 次元, 8 次元のデータを使用する．$k = 5, 10, 20$ とし，それぞれの k で $k' = 400, 600, 800, 1000$ として，各次元の一括生成ファイルより k-最近傍点を求め，解答の F 値(注2)を計測する．これを質問点に従い 100 回行い，F 値の平均を算出する．また，100 回の検索に要するページアクセス回数の合計を計測する．

F 値を表 1 に，ページアクセス回数を表 2 に示す．表 2 のページアクセス回数については，k の値に依らないことに注意する．

5.3 実　験　2

実験 2 では次元縮小を用いた EGF における逆最近傍検索の精度と検索効率について実験を行う．各データにより作成した一括生成ファイルにおいて 100 回の逆最近傍検索を行い，解答の F 値とページアクセス回数を計測する．逆最近傍検索中の k/k'-最近傍検索における k, k' の値は，実験 1 と同様の値で

(注1)：本実験では `www.rtreeportal.org` で公開されている地図情報を用いた

(注2)：A=質問 q の解の集合，R=本来の解の集合として，再現率 $recall = \frac{|A \cap R|}{|R|}$, 適合率 $precision = \frac{|A \cap R|}{|A|}$, $F = \frac{2 * recall * precision}{recall + precision}$

表 1　実験 1 F 値

k'	400			600			800			1000		
k	5	10	20	5	10	20	5	10	20	5	10	20
4-dim	0.51	0.52	0.50	0.67	0.68	0.67	0.80	0.81	0.79	0.88	0.88	0.87
6-dim	0.58	0.57	0.56	0.76	0.76	0.74	0.87	0.87	0.86	0.93	0.94	0.93
8-dim	0.67	0.66	0.63	0.80	0.80	0.77	0.88	0.88	0.88	0.94	0.94	0.94

表 2　実験 1 ページアクセス回数

k'	400	600	800	1000
4-dim	39820	58306	76072	93465
6-dim	40485	58967	77025	95000
8-dim	41407	59838	77999	96253

行う.

重心数 100, 239, 100000 でのそれぞれの F 値の平均とページアクセス回数の合計を表 3 に示す．また，重心数 100 での 100 回の逆最近傍検索に要する時間を表 4 に示す.

5.4 実　験　3

実験 3 では，次元縮小を用いた EGF の動的特性について実験を行う．それぞれの一括生成ファイルに追加投入を行い，その後 100 回の逆最近傍検索を実行し，解答の F 値とページアクセス回数の合計を計測する．逆最近傍検索中の k/k'-最近傍検索における k, k' の値は，実験 1 と同様の値で行う.

重心数 100, 239, 100000 でのそれぞれの F 値の平均とページアクセス回数の合計を表 5 に示す.

5.5 考　　察

各実験についての考察を行う.

実験 1 では，高い精度となる k' でも高速な k/k'-最近傍検索を行うことができることを示す．表 1 より，各次元の 5-最近傍検索, 10-最近傍検索, 20-最近傍検索がほぼ同様の曲線となっている．これは，距離の誤差が次元縮小法にのみ起因するためである．この結果より約 0.9 の F 値を得ることができる k' の値は，4 次元で 1000, 6 次元で 800, 8 次元で 600 であり，k' の値は $k' = \frac{\text{元の次元}}{\text{縮小次元}} * 40$ 程度で十分であると考える．このときのページアクセス回数はそれぞれ，4 次元で 93465 回, 6 次元で 77025 回, 8 次元で 59838 回であり，線形走査での k-最近傍検索のページアクセス回数 500000 回と比較するとそれぞれ約 5.4 倍, 6.5 倍, 8.4 倍の検索速度となる.

表 6　実験 1 元データページへのアクセス回数

k'	400	600	800	1000
4-dim	38462	56661	74081	91007
6-dim	38519	56717	74227	91135
8-dim	38505	56679	74244	91100

また，表 6 はページアクセス回数中の元データのページへのアクセス回数を示す．これよりページアクセスのほとんどが元の次元のデータページへのアクセスであり，次元縮小に伴う距離の誤差が小さくなれば，さらに速度を向上させることができる.

実験 2 は，次元縮小を用いた EGF における逆最近傍検索が，高い精度で効率よく実行できることを示す.

まず精度と検索効率について考察を行う．ここでは重心数 100, k=10 での実験結果で考察を行う.

F 値が約 0.9 となる k' の値での，各次元での F 値はそれぞれ 4 次元で 0.89, 6 次元で 0.88, 8 次元 0.72 であり，解答は十分に信頼できると考える．このときのページアクセス回数はそれぞれ 4 次元で 150250 回, 6 次元で 110860 回, 8 次元で 92635 回, 実時間は 4 次元で 45.02 秒, 6 次元で 48.37 秒, 8 次元で 468.00 秒である．線形走査での逆最近傍検索のページアクセス回数が 500000000 回であることを考慮すると非常に高速な検索を行うことができる．文献 [1] では，21000 件のデータで R 木を用いて逆最近傍検索を行っており，その平均ページアクセス回数と平均時間は，約 260 回と約 100 秒である．本稿での実験環境に合わせると，100 回の逆最近傍検索にかかるページアクセス回数は約 130000 回，実時間は 50000 秒となる．ページアクセス回数では 4 次元で 0.87 倍, 6 次元で 1.2 倍, 8 次元で 1.4 倍となりほぼ同様の結果となるが，実時間では 4 次元で 1110 倍, 6 次元で 1033 倍, 8 次元で 106 倍となり，高速な検索であることが分かる．これは，EGF による高い検索効率が次元縮小を用いても損なわないことを示す．また k-最近傍検索と同様に，ページアクセスのほとんどが元の次元のデータページへのアクセスであるため，次元縮小に伴う距離の誤差が小さくなれば，さらに速度を向上させることができる.

次に各変数による F 値，ページアクセス回数の変化について考察する.

重心数の違いによる F 値，ページアクセス回数の変化はほとんどない．(b) でのページアクセス回数が他よりやや高いが，誤差の範囲であると考える．これより，Centroid SVD により SVD とほぼ同じ性能の行列 U が作成できることが分かる.

k-最近傍検索とは違い，逆最近傍検索では k の値が増加するとページアクセス回数は変化せずに F 値のみが向上する．これは，k の増加に伴い再現率が上昇するが，フィルタリング処理により適合率を落とさないからであり，このフィルタリング処理がうまく動作していることが分かる．ただし，5 から 10 での F 値の変化に比べて，10 から 20 での F 値の変化は小さいので，k の値は 10 で十分であると考える.

k' の値が増加することは検索範囲を拡大することと同様であり，F 値は向上するがページアクセス回数も増加する．実験 1 で約 0.9 の F 値を得る k' の値では本実験でも 0.9 程度の F 値が観測できることから，逆最近傍検索でも $k' = \frac{\text{元の次元}}{\text{縮小次元}} * 40$ 程度で十分であると考える．次元数の増加により，F 値が向上し，かつページアクセス回数が減少している.

ページアクセス回数の減少の要因として，次元が高いと距離誤差が小さくなり，ブール範囲検索処理時に調べる無関係なデータの数が減ることが挙げられる．ただし表 4 から分かるように次元が増加するとディレクトリが大きくなり，メモリ内での計算時間，すなわちシステム時間が大きくなるので，実際には 4 次元から 6 次元のほうが適している.

実験 3 は，次元縮小を用いた EGF の高い動的特性を示して

表 3　実験 2 F 値とページアクセス回数

k'			400			600			800			1000		
k			5	10	20	5	10	20	5	10	20	5	10	20
重心数 100	F 値	4-dim	0.48	0.52	0.52	0.60	0.64	0.65	0.73	0.82	0.84	0.77	0.89	0.90
		6-dim	0.56	0.59	0.60	0.69	0.73	0.74	0.78	0.88	0.90	0.84	0.94	0.97
		8-dim	0.56	0.60	0.61	0.66	0.72	0.74	0.78	0.87	0.90	0.82	0.92	0.96
	ページアクセス回数	4-dim	88024	90391	89096	108115	109996	109516	131613	135258	135629	146567	150250	150491
		6-dim	80081	80337	79578	96020	96375	96543	109901	110861	111283	120478	122261	122860
		8-dim	80037	81669	79970	91708	92635	93133	104323	105244	105625	117048	118081	118283
重心数 239	F 値	4-dim	0.53	0.57	0.58	0.61	0.65	0.65	0.76	0.85	0.85	0.82	0.92	0.96
		6-dim	0.60	0.63	0.64	0.72	0.77	0.78	0.81	0.88	0.92	0.83	0.93	0.97
		8-dim	0.64	0.62	0.63	0.69	0.76	0.78	0.79	0.86	0.88	0.84	0.94	0.98
	ページアクセス回数	4-dim	105530	110374	109316	124301	127558	129312	146929	152090	154130	164157	170483	177014
		6-dim	94401	96318	96097	109350	111670	110806	121653	124637	127459	131877	135127	138713
		8-dim	83146	83713	83146	94500	96794	96814	105971	107865	108663	119091	121356	123268
重心数 100000	F 値	4-dim	0.43	0.47	0.47	0.58	0.64	0.63	0.76	0.85	0.87	0.83	0.93	0.95
		6-dim	0.54	0.57	0.58	0.66	0.74	0.76	0.78	0.86	0.88	0.81	0.90	0.93
		8-dim	0.64	0.67	0.68	0.70	0.78	0.80	0.80	0.88	0.90	0.90	0.91	0.95
	ページアクセス回数	4-dim	91534	93764	93234	111584	114784	115509	139437	142560	143133	154942	158593	158385
		6-dim	87401	89251	88453	106133	107133	106999	118827	119480	119890	130671	132570	132559
		8-dim	85611	87030	86819	97426	99076	99615	111410	112409	112587	122932	124730	124802

表 4　実験 2 重心数 100 での実行時間 [sec]

k'		400			600			800			1000		
	k	ST	UT	RT	ST	UT	RT	ST	UT	RT	ST	UT	RT
4-dim	5	7.22	2.16	26.39	10.98	2.75	31.47	15.88	3.61	37.64	19.98	3.97	42.61
	10	8.05	2.35	27.86	11.83	3.18	32.92	17.38	4.08	39.74	21.63	4.69	45.02
	20	7.85	2.59	27.69	12.16	3.22	33.28	18.18	4.31	40.79	22.42	4.93	46.31
6-dim	5	9.72	2.21	30.99	15.56	2.68	37.81	21.51	3.37	45.19	27.00	3.63	51.45
	10	11.83	2.52	33.32	18.44	3.13	41.20	24.43	3.77	48.37	31.69	4.22	56.73
	20	12.04	2.72	33.70	18.27	3.55	41.38	24.94	3.83	49.52	31.35	4.37	56.55
8-dim	5	232.25	3.20	0.00	354.73	3.66	382.16	617.77	3.99	646.52	756.87	4.63	787.39
	10	286.59	3.70	313.40	439.46	4.53	468.00	812.82	5.12	842.70	988.42	5.77	1020.00
	20	318.72	3.72	345.50	469.16	4.52	497.64	801.97	6.47	837.78	971.44	5.64	1002.90

ST:システム時間 UT:ユーザ時間 RT:実時間

いる．

実験 3 で得られる結果を実験 2 のいずれの結果と比較しても，大きくとも F 値は 0.1 程度の減少, アクセスコストは 20000 回程度の増加であり，更新後も高い精度と検索効率を保持することが分かる．特に実験 1 で F 値が 0.9 を超える重心数 100, k=10, k'=1000(4 次元), 800(6 次元), 600(8 次元) での F 値は, それぞれ 4 次元で 0.85, 6 次元で 0.81, 8 次元で 0.71 である．実験 2 と比較するとそれぞれ 0.04, 0.07, 0.01 の減少であり，精度はほぼ変化しないことが分かる．このときのページアクセス回数はそれぞれ 4 次元で 164262 回, 6 次元で 123424 回, 8 次元で 105245 回である．実験 2 と比較するとそれぞれ 9.3%, 11.3%, 13.6%の増加であり，20%のデータ増加であることを考えると小さな増加である．

6. むすび

本論文では，次元縮小を用いた拡張可能グリッドファイルによる逆最近傍検索手法の精度とアクセスコストを実験により評価し，その有用性を提示した．また，Centroid SVD より生成した射影行列 U の耐えうる範囲ではデータを更新しても性能がほとんど落ちないことを示した．今後の課題として，次元縮小時の距離誤差をさらに小さくする次元縮小法の適応が挙げられる．

謝　辞

本研究の一部は文部科学省科学研究費補助金 (C) (課題番号 16500070) の支援をいただいた．

文　献

[1] Amit,S., Hakan,F. et al.: "High Dimensional Reverse Nearest Neighbor Queries", ACM *CIKM* , 2003, p.91-98

[2] Beckmann, N. et al.: "The R*-tree : An Efficient and Robust Access Method for Points and Rectangles", *SIGMOD* 1990, p.322-331

[3] C.Bohm, S.Berchtold, et al.: "Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases", *ACM Comp. Surveys* 33-3, pp.322-373, 2001

[4] Freeston, M.: "The BANG file – A New Kind of Grid File",

表 5　実験 3 F 値とページアクセス回数

k'			400			600			800			1000		
k			5	10	20	5	10	20	5	10	20	5	10	20
重心数 100	F 値	4-dim	0.45	0.48	0.48	0.56	0.58	0.59	0.70	0.75	0.76	0.76	0.85	0.86
		6-dim	0.56	0.57	0.58	0.64	0.67	0.68	0.75	0.81	0.82	0.82	0.88	0.91
		8-dim	0.56	0.58	0.59	0.67	0.71	0.72	0.80	0.85	0.86	0.83	0.90	0.92
	ページアクセス回数	4-dim	94371	97702	95261	114108	115100	113755	141261	143649	143743	157020	164262	163573
		6-dim	87660	88955	87700	101117	102981	102310	121608	123424	123038	132480	134274	135448
		8-dim	91020	92273	91396	103063	105245	104146	117464	119267	118969	127538	129788	129650
重心数 239	F 値	4-dim	0.43	0.46	0.45	0.56	0.61	0.62	0.71	0.76	0.77	0.76	0.83	0.84
		6-dim	0.49	0.51	0.51	0.69	0.73	0.73	0.76	0.83	0.83	0.85	0.92	0.93
		8-dim	0.53	0.56	0.57	0.67	0.71	0.73	0.76	0.81	0.82	0.83	0.88	0.92
	ページアクセス回数	4-dim	102727	104691	103134	128360	137506	137644	154844	162910	164061	168802	179208	180430
		6-dim	95573	98563	98463	120947	123717	123872	132547	139112	150690	144270	150846	151063
		8-dim	91690	93742	93907	107279	109744	109316	118204	121083	121195	129607	131927	133148
重心数 100000	F 値	4-dim	0.41	0.43	0.43	0.53	0.53	0.55	0.69	0.73	0.75	0.83	0.88	0.90
		6-dim	0.49	0.51	0.52	0.71	0.75	0.77	0.78	0.84	0.85	0.86	0.92	0.94
		8-dim	0.59	0.64	0.65	0.73	0.78	0.80	0.79	0.85	0.86	0.88	0.95	0.96
	ページアクセス回数	4-dim	97712	100309	99384	118269	122104	121676	143646	149064	149641	171723	179680	181381
		6-dim	87980	90195	88978	116609	119978	119931	130957	134456	134188	142409	145786	145609
		8-dim	89798	92598	91477	107845	111955	111830	121310	124728	124614	135302	138377	138577

proc.*SIGMOD* 1987, p.260-269

[5] Gaede, V. and Gunther, O.: "Multidimensional Access Methods", *ACM Comp. Surveys* 30-2, pp.170-231, 1998

[6] A. Guttman: "R-trees: A dynamic index structure for spatial searching" , proc. Int. Conf. ACM SIGMOD '84, pp.47-57,1984

[7] T.R. ハーブロン, 遠山元道 (訳): "ファイルシステム" , 啓学出版,1992

[8] G. Hjaltason, H. Samet: "Distance Browsing in Spatial Databases" , ACM TODS. vol. 24(2), pp.265-318,1999

[9] J.Gao , Jun Zhang: "Text Retrieval Using Sparsified Concept Decomposition", CIS, 2004, Shanghai

[10] K.V.R. Kanth et al.: "Dimensionality Reduction for Similarity Searching in Dynamic Databases" , proc.*SIGMOD* 1998, pp.166-176

[11] 片山 紀夫, 佐藤 真一: "マルチメディア情報の大規模処理に向けた多次元インデクシング手法の応用" , 電子情報通信学会論文誌 (D-II),vol.J82-D-II,no.10,pp.1606-1616,Oct,1999

[12] F.Korn, S.Muthukrishnan et al.: "Influence Sets Based on Reverse Nearest Neighbor Queries", proc.*SIGMOD* , 2000, p.201-212

[13] 北 研二, 津田 和彦, 獅子堀 正幹: "情報検索アルゴリズム", 共立出版, 2002.

[14] 三好 涼介, 三浦 孝夫, 塩谷 勇"拡張可能グリッドファイルにおける最近傍検索の改善" , 電子情報通信学会論文誌 (D-1),Vol.J88-D-I,No.3,pp.727-734,2005

[15] Miyoshi,R., Miura,T., et al.: "Nearest Neighbor Queries on Extensible Grid Files Using Dimensionality Reduction", IEEE COMPSAC , 2005, p.249-255

[16] J. Nievergelt , H, Hinterberger: "The Grid File: An Adaptable, Symmetric Multikey File Structure" , ACM TODS. vol. 9, pp.38-71,Mar.1984

[17] 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 (D-I),Vol.J74-D-I,No.8,pp.467-475,1991

[18] 坂内正夫, 大沢裕: "画像データベース" , 昭晃堂,1987

[19] Papadimitriou, C. H., Raghavan, P., Tamaki, H. and Vempala, S.: "Latent semantic indexing: A probabilistic analysis", In *Proc. 17th ACM Symp. on the Principles of Database Systems*, pp 159-168, 1998.

[20] N. Roussopoulos , et al.: "Nearest Neighbor Queries" , proc.*SIGMOD* 1995, p.71-79

[21] Sakurai,Y., Yoshikawa, M. et al.: "The A-tree : An Index Structure for High-Dimensional Spaces Using Relative Approximation", proc.*VLDB* 2000, p.516-526

[22] I.Stanoi, D.Agrawal et al.: "Reverse Nearest Neighbor Queries for Dynamic Databases", *ACM SIGMOD* Workshop on DMKD , 2000, p.44-53

[23] Weber,R., Schek. H.J. et al: "A Quantitative Analysis and Performance Study for Similarity-Search Methods in High Dimensional Spaces", proc.*VLDB*, 1998, p.194-205

[24] White, D.A. et al.: "Similarity Indexing with the SS-tree", IEEE *ICDE*, 1996, p.516-523

[25] C.Yang, K.-I.Lin: "An Index Structure for Efficient Reverse Nearest Neighbor Queries", ICDE , 2001, p.485-492