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　現在の高次元データを扱う索引手法では，次元が高くなるにつれて任意のデータ間の距離が大きくなり，最近傍検索においてはある次元を超えるとすべてのページにアクセスしなければならなくなる．その結果，全件走査の性能に劣ることが知られている (いわゆる ”次元の呪い”)．本稿では，データの次元を縮小し，そのデータに対して拡張可能グリッドファイルを用いて索引付けを行う．これにより，高次元に起因する更新および検索コストを低減し，次元縮小による誤差の拡大を吸収できることを示す.

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

Querying 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　Nowadays, when we examine nearest neighbor search by multi-dimensional indexing structure, we must access all pages if dimensionality exceeds a certain threshold.This is because a distance between the optional data is longer as dimensional is higher , as a result, all indexing structures outperformed by sequential scan–known as ”curse of dimension”. In this investigation, we propose a sophisticated access mechanism based on Dimensionality Reduction and Extensible Grid Files.It can solves the problem which originate high dimension, and examine nearest neighbor search contains error region.

Key words　Multi-dimensional data processing system , Extensible Grid Files , Dimensionality Reduction

1. 前　書　き

近年のデータベースにおいては，医療や天文学などの高次元データを使用する分野が少なくない．現在，高次元データを処理するにあたり，SS 木や SR 木などの多次元索引構造 [3], [5], [13] を用いることが多い.

R 木 [4] は，空間データ集合を格納すべき最小包囲長方形 (Minimum Bounding Rectangle, MBR) と呼ばれる超長方形を領域分割方式に従って再帰的，階層的に分割する多分木である．従来の階層索引構造と違い MBR の重なりを許容するため平衡木となる．しかし重なりを許容することで，質問によってはトラバースが発生したり，動的挿入，更新時の処理負荷が増加するなどの問題がある.

SR 木 [7] は最小包囲球 (Minimum Bounding Sphere, MBS) と MBR を組み合わせ，重なる部分をノードとしたものである．径も体積も小さくなり検索に優れている．しかし R 木と同じようにノードの重なりを許容するため，平衡木にはなるがトラバースの発生やデータ更新効率の悪化などの問題がある.

拡張可能グリッドファイル [10](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.

これらの多次元索引構造による高次元データの検索では，次元が高くなるにつれて任意のデータ間の距離が大きくなり，最近傍検索においてはある次元を超えるとすべてのページにアク

セスしなければならなくなる．その結果，全件走査の性能に劣ることが文献[18]において報告されている(いわゆる”次元の呪い”)．

しかし，文献[18]ではデータの一様分布を仮定して解明を行っている．現実には一様に分布するデータはほとんど存在せず，ましてや高次元空間で各次元に関して一様であるデータはありえない．この意味で，”次元の呪い”が現実の高次元データにおいてどこまで影響を持つかは解明されていない．

本稿では，偏りを持つ高次元データでは多次元索引構造で十分に対応できることを実験により証明する．また次元縮小法を用いて高次元データの次元を縮小し，そのデータに対して拡張可能グリッドファイルを用いて索引付けを行う．これにより，高次元に起因する検索コストをさらに低減し，次元縮小による誤差の拡大を吸収できることを示す．

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，データ領域を拡張する．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せねばならない．このため，管理領域そのものの桁数p_nを記憶する必要がある．

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．データを移動させたあと，前のバケットに管理されていた領域の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，P_n-p_nが同じならばデータ分布の大きい軸を，それ以外ならP_n-p_nが最小の軸を分割軸とする．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，最小のP_nを持つ軸が複数存在するなら第1次元に最も近い軸を選択する．

[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．なお，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，それ以外はエリミネイトリストに加えておく．すべての索引の判定終了後，アクセスリス

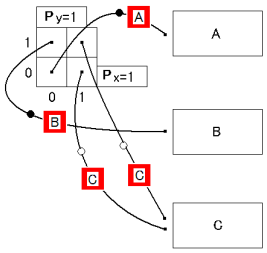


図 1　拡張可能グリッドファイル
Fig. 1　Extensible Grid File

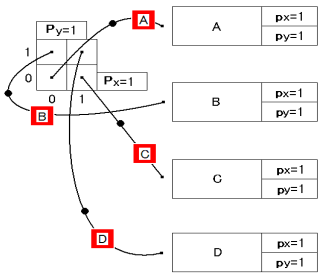


図 2　バケット C 分割後のキー分布
Fig. 2　Dividing bucket C

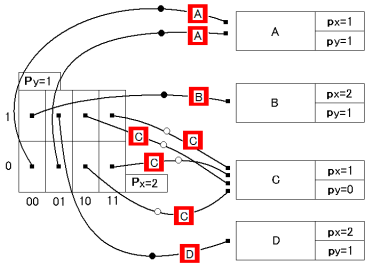


図 3　ディレクトリ拡張後のキー分布
Fig. 3　Doubling directory

トにあるバケットのデータに対して実際の判定を行う．

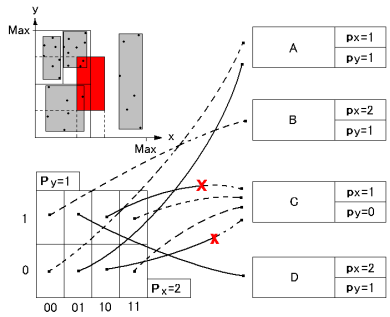


図 4　範 囲 検 索

2.3.3　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を求める操作を最近傍検索という [15].

検索点のそれぞれ上位 P_n 桁を取り出し索引とする．取り出した索引において判定を行い，内在判定が真なら索引の対応するバケット内のデータで距離計算を行い仮の最近点 N を算出する．内在判定が偽なら，指定された索引の周囲の索引に対応するバケットにアクセスし，データが存在するバケットを検出するまで索引の検索範囲を拡げ，検出されたバケットの中から仮の最近点 N を算出する．N 算出後，質問点と N との距離を L とし，質問点を中心とした辺長 $2L$ の正方形領域に対して範囲検索を行う．このとき，N を算出する際に調べた索引は検索対象から排除する．得られたデータより真の最近点 N' を算出する．

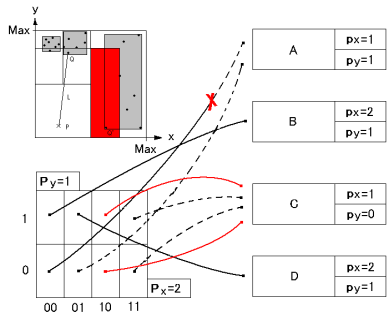


図 5　最近傍検索

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 10 次元程度の低次元データでは効率よく動作するが，それ以上の次元を有するデータに対する処理は効率の悪いものとなり，最近傍検索においては全件走査の処理に劣る性能となる．

これは，次元が高くなると任意のデータ間の距離が大きくなり，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である．例えば，$q_n = (1, 1, \cdots, 1)$ に対して仮の最近点が $a_n = (0, 0, \cdots, 0)$ である場合を考える．2 次元の場合，真の最近点算出での検索範囲は辺長 $\sqrt{2}$ の正方形領域であるが，100 次元の場合，その検索範囲は辺長 10 の超立方体領域となる．この高次元に起因する問題は，”次元の呪い (Curse of Dimensionality)” と呼ばれる．

本稿では，次元縮小を用いて高次元データの次元を縮小する索引構造を提案する．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。ここで、後々次元縮小データで検索することを考慮すると扱う次元縮小法は近似誤差をなるべく小さくするものが良い。よって本稿では，特異値分解を用いた次元縮小法を使用し、本章でこの次元縮小法について述べる．

3.1　Latent Semantic Indexing

Latent Semantic Indexing（潜在的意味索引付け，LSI）[8], [14] 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 を考える．行列の大きさは d 行 N 列であり，それぞれの列ベクトルが 1 件のデータを表している．データ行列 X の SVD は，次の式で表される．

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

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，$S_{11} \geq S_{22} \geq \cdots$ 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 k 次元 ($k << d$) に縮小する．LSI における次元縮小は，次の計算で行う．

$$X^{SVD}_{k\times N} = U^T_k X \tag{2}$$

U_k は大きさ $(d \times k)$ の行列で，行列 U から最初の k 個の左特異ベクトルを抜き出したものである．k 個の左特異ベクトルは，最も大きな k 個の特異値に対応している．

ここで，データの次元縮小に伴い，誤差が生じる．次元縮小後のデータ行列 X_k が元のデータ行列 X をどの程度近似しているかを測るための尺度としてフロベニウス・ノルムを用いる．$d \times N$ 行列 X におけるフロベニウス・ノルム $||X||_F$ は，次の式で定義される．

$$||X||_F = \sqrt{\sum_{i=1}^{d}\sum_{j=1}^{N} x_{ij}^2} \tag{3}$$

この近似誤差は，特異値によって算出できることが知られている．よって，X と X_k の違い $Diff$ は，以下の式で算出できる．

$$Diff = \frac{||X - X_k||_F}{||X||_F} = \frac{\sqrt{S_{k+1k+1}^2 + \cdots + S_{rr}^2}}{\sqrt{S_{11}^2 + S_{22}^2 + \cdots + S_{rr}^2}} \tag{4}$$

データ検索を行う場合は，データ行列 X を次元縮小するために算出した射影行列を使用して，検索点を縮小次元空間に射影し検索を行う．

$$\mathbf{q}_{k\times 1}^{SVD} = U_k^T \mathbf{q}_{d\times 1} \tag{5}$$

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 [16] 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 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_k 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．

$$\mathbf{a}_{k\times 1}^{SVD} = U_k^T \mathbf{a}_{d\times 1} \tag{6}$$

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，次元縮小データに対する拡張可能グリッドファイルの検索手法を論じる．ここでは，n 次元のデータを k 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_k 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 (4) の $Diff$ を用いて以下で表すことができる．

$$LOW_i = Q_i' - \frac{r}{2} \cdot (1 + Diff)(i = 1, \cdots, k)$$
$$HIGH_i = Q_i' + \frac{r}{2} \cdot (1 + Diff)(i = 1, \cdots, k)$$

この 2 頂点のそれぞれ上位 P_k 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，それ以外はエリミネイトリストに加えておく．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内の縮小次元データから $LOW_i \leqq D_i' \leqq HIGH_i (i = 1, \cdots, k)$ 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．

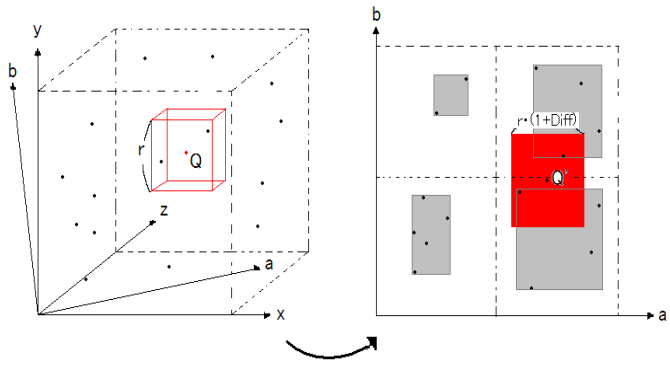


図 6　次元縮小後の範囲検索

4.3 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，検索点 Q' のそれぞれ上位 P_n 桁を取り出し索引とする．取り出した索引において判定を行い，内在判定が真なら索引の対応するバケット内のデータで距離計算を行い仮の最近点 N' を算出する．内在判定が偽なら，指定された索引の周囲の索引に対応するバケットにアクセスし，データが存在するバケットを検出するまで索引の検索範囲を

拡げ，検出されたバケットの中から仮の最近点 N' を算出する．ここで，縮小次元空間における N' と Q' の距離 L' より，元の空間における N と Q の距離 L は式（4）の $Diff$ により $L' \cdot (1-Diff) \leqq L \leqq L' \cdot (1+Diff)$ の範囲内の値をとる．よって，検索点 Q' を中心とした辺長 $2L' \cdot (1+Diff)$ の正方形領域に対して範囲検索を行い，$LOW_i \leqq D'_i \leqq HIGH_i (i = 1, \cdots, k)$ を満たす縮小次元データを算出する．ただし，ここでの範囲検索は縮小次元空間での範囲検索であるので，LOW,$HIGH$ の値はさらに誤差を含み，

$$LOW_i = Q'_i - L' \cdot (1 + Diff)^2 (i = 1, \cdots, k)$$
$$HIGH_i = Q'_i + L' \cdot (1 + Diff)^2 (i = 1, \cdots, k)$$

となることに注意する．このとき，Q を算出する際に調べた索引は検索対象から排除する．得られるデータの元データより，真の最近点 N を算出する．

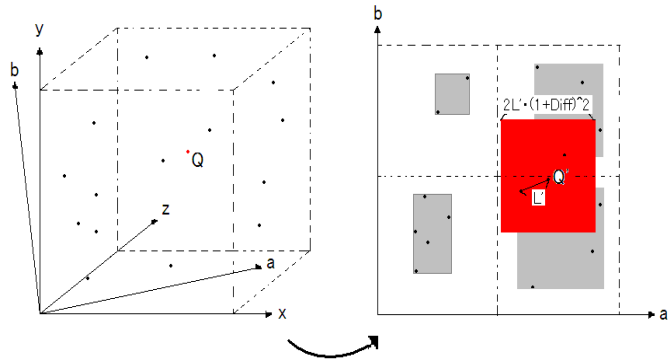


図 7　次元縮小後の最近傍検索

5. 性能評価

本章では，高次元データに対する多次元索引構造の性能を，EGF を用いて実験により評価する．また次元縮小を用いた EGF 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 点と 100 次元の一様分布データ 120,000 点を使用する．非一様分布データはニューヨーク，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+19898 点に分割し初期生成 (静的挿入)，追加挿入 (動的投入) のそれぞれで性能を調べる．また，質問点 100 点はそれぞれの分布に従い生成する．1 バケットを 1 ページとし，$B = 32$ Kbyte として実験を行う．

5.2 実験 1

100 次元の点データ 100000 点を k-mean クラスタリング手法を用いて 100，200，400 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(注2)．その後，行列 U より 4 次元,8 次元,16 次元,32 次元に次元縮小する．そのデータに対して一括生成を行い最近傍検索を実行し，各次元での正解率を計測する．正解率は，最近傍検索を実行したときにその結果が真の最近点であれば正解とし，これを 100 回実行したときの正解数の割合である．また，検索でのページアクセス回数の計測を行い，次元縮小を用いない EGF, 全件走査と比較を行う．正解率を図 8 に，ページアクセス回数を図 9 示す．図 9 は 100 回の問合せにかかるページアクセス回数の合計であり，図中 100-mean,200-mean,400-mean はそれぞれの重心の数より求めた行列を用いて次元縮小を行った EGF を表し，EGF は次元縮小を用いない EGF を表す．

5.3 実験 2

残りの 100 次元の点データ 20000 点を，先に求めた行列 U を用いて 4 次元,8 次元,16 次元,32 次元に次元縮小し，そのデータを実験 1 で構築したそれぞれの一括生成ファイルに投入する．その後最近傍検索を実行し，各次元での正解率を計測する．各検索のページアクセス回数の計測を行う．また実験 1 と同様に，最近傍検索でのページアクセス回数の計測を行い，次元縮小を用いない EGF, 全件走査と比較を行う．正解率を図 10 に，ページアクセス回数を図 11 示す．図 11 は 100 回の問合せにかかるページアクセス回数の合計である．

5.4 考察

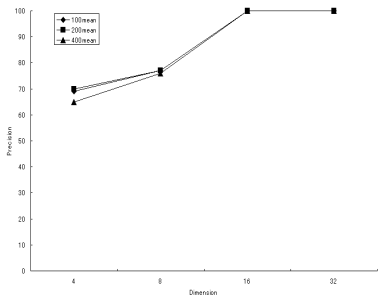
各実験についての考察を行う．まず次元縮小を用いない EGF と全件走査との比較より，高次元データにおける”次元の呪い”の影響を考察する．

実験では，多次元索引構造で十分に対応できることを示している．初期生成後の非一様分布データでは，次元縮小を用いない EGF は全件走査と比べて約 2.1 倍のアクセスコスト比となり，追加挿入後では約 2.5 倍のアクセスコスト比となる．これは，全件走査の性能がデータ数に比例して悪化するためであり，データ数が大きくなると検索コストの差はさらに大きくなる．また一様分布データにおいて，初期生成後の次元縮小を用いない EGF の検索性能は全件走査と比べて約 0.6 倍のアクセスコスト比となる．しかし追加挿入後では約 0.7 倍のアクセスコスト比となり，差異が小さくなっている．これも非一様分布データでの現象と同様，全件走査の性能がデータ数に比例して悪化することに起因し，データ数が大きくなると検索コストの差はさらに小さくなる．

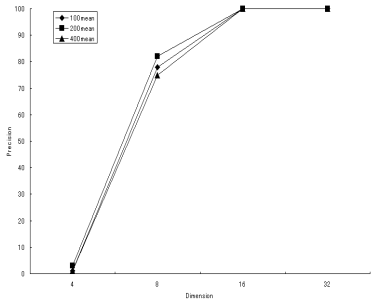
次に，次元縮小を用いた EGF の検索性能について考察を行う．実験 1 では，4 次元から 8 次元において，高い正解率を保持し，かつ次元縮小によりアクセスコストを低減できることが確認できる．

非一様分布データでは，4 次元，8 次元において次元縮小の効果がみられ，4 次元では次元縮小を用いない EGF，全件走査と比べてそれぞれ最大で重心数 200 のとき約 57 倍，約 117 倍のアクセスコスト比となり，8 次元ではそれぞれ最大で重心数 200 のとき約 1.8 倍，約 3.8 倍のアクセスコスト比となる．ただし正解率は 4 次元では最大で重心数 200 のとき 70%，8 次元では最大で重心数 100,200 のとき 77%となり，4 次元での検索性能は 8 次元での検索性能に比べ多少精度の低い回答となる．また，16 次元，32 次元では正解率は 100%となるが，検索コス

(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)：特異値分解には `GNU Octave 2.1.36` を使用した

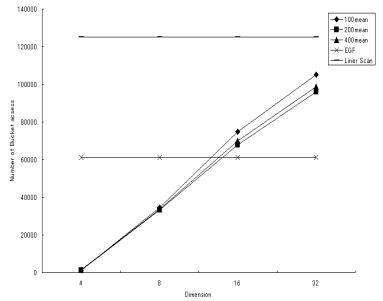


(a) 非一様データ

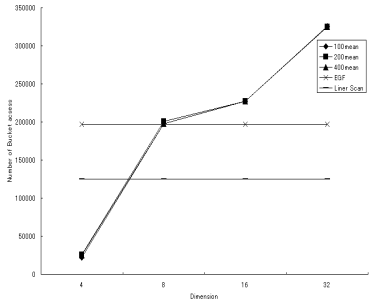


(b) 一様データ

図 8　実験 1 正解率

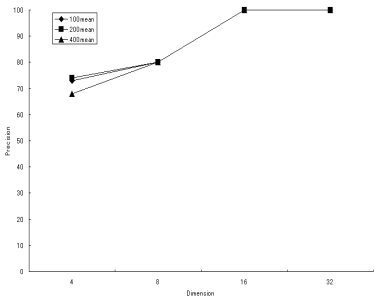


(a) 非一様データ

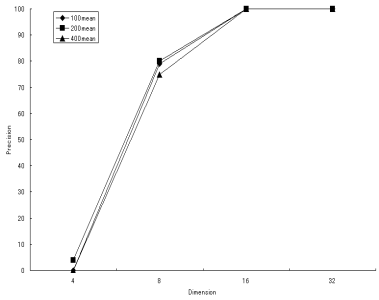


(b) 一様データ

図 9　実験 1 ページアクセス回数

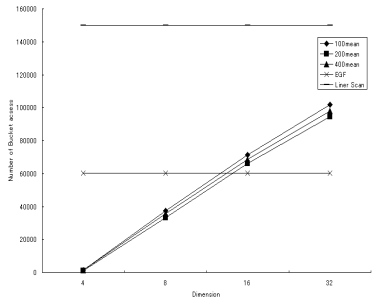


(a) 非一様データ

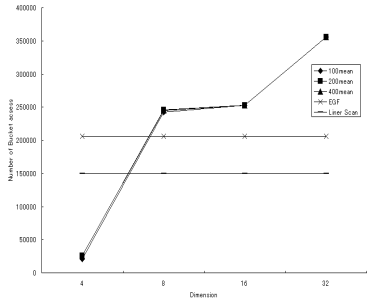


(b) 一様データ

図 10　実験 2 正解率



(a) 非一様データ



(b) 一様データ

図 11　実験 2 ページアクセス回数

トにおいて次元縮小を用いない EGF，全件走査と比べて劣る性能となる．この次元数では，高次元に起因する問題が影響することを示している．

一様分布データでは，4 次元においてアクセスコストは低くなるが，正解率が最大でも 3%となり有益な検索ではない．8 次元では，正解率は最大で 82%となるがアクセスコストが次元縮小を用いない EGF と同等，全件走査より劣る性能となる．16 次元以上では，非一様分布データと同じように正解率は 100%となるが，高次元に起因する問題が影響し，次元縮小を用いない EGF，全件走査より高価な検索コストとなる．これより，4 次元では LSI による誤差保障の範囲を高確率で超えるため正解率が低く，8 次元以上では誤差保障の範囲が大きくなり，結果として全件走査より高いコストとなることが考察できる．以上より，4 から 8 次元の範囲に正解率を高く維持し，より安価なア

クセスコストで検索を行える最適な縮小次元が存在すると推測できる．

実験 2 では，Centroid SVD による次元縮小が，EGF の高い動的特性を損なわないことが確認できる．

非一様分布データでは，全件走査が 25000 のアクセスコスト増加に比べて，EGF を使用する場合は実験 1 と変わらないアクセスコストとなる．4 次元では次元縮小を用いない EGF，全件走査と比べてそれぞれ最大で重心数 200 のとき約 54 倍，約 134 倍のアクセスコスト比となり，8 次元ではそれぞれ最大で重心数 200 のとき約 1.8 倍，約 4.5 倍のアクセスコスト比となる．これより次元縮小を用いる場合でも EGF は高い動的特性を保持することが確認できる．また正解率も実験 1 に比べて 3%から 4%向上し，4 次元では最大で重心数 200 のとき 74%，8 次元ではすべての重心数で 80%となる．これより Centroid SVD より多少の更新にも耐えうる射影行列を算出できる．ただし 16 次元以上では，実験 1 と同様に高次元に起因する問題が影響し，次元縮小を用いない EGF，全件走査より高価な検索コストとなる．

一様分布データでは，4 次元において実験 1 と変わらないアクセスコストとなる．実験 1 の考察より最適な縮小次元は 4 から 8 次元であると考え，その次元においても高い動的特性は保持できる．

6. 結　　論

本論文では，高次元データにおける多次元索引構造の性能を実験により評価し，現実に存在する非一様分布データにおいては多次元索引構造で対応できることを実証した．高次元データが一様分布になるにつれ”次元の呪い”の影響を受けることが実験からも判明したが，同時に高い確度で多次元索引構造でも処理できることを示した．また次元縮小を用いた EGF を定義し，その検索手法の正解率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．今後の課題として，膨大なデータ量を想定しひとつの計算機のみに特化せず，複数の計算機による分散環境下での EGF の構築などが挙げられる．

謝　　辞

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

文　　献

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

[2] Freeston, M.: The BANG file – A New Kind of Grid File, proc.*SIGMOD* 1987, p.260-269

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

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

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

[6] 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

[7] 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

[8] 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.

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

[10] 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 (D1), 2005 3 月予定

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

[12] 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

[13] 坂内正夫, 大沢裕: ”画像データベース” , 昭晃堂,1987

[14] 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.

[15] N. Roussopoulos , et al ”Nearest Neighbor Queries” , proc.*SIGMOD* 1995, p.71-79

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

[17] 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

[18] 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

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