社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

空間インデクシングに基づく距離推定を用いた高速かつ省メモリな近似最近傍探索

佐藤　智一† 　岩村　雅一† 　黄瀬　浩一†

† 大阪府立大学大学院工学研究科　〒 599–8531 堺市中区学園町 1–1
E-mail: †sato@m.cs.osakafu-u.ac.jp, ††{masa,kise}@cs.osakafu-u.ac.jp

あらまし　本稿では，高次元かつ大規模なデータセットを高速かつ省メモリで扱うための近似最近傍探索手法を提案する．近似最近傍探索は，入力ベクトルに最も距離が近いベクトルを探索する最近傍探索問題において，探索誤りを許容することで計算時間，メモリ使用量を大幅に削減するものである．ハッシュ構造を用いてクエリからの距離を推定する手法に改良を加え，より一般的なデータに対して高速に解を得られるようになった．また，ベクトルを量子化表現することでメモリ使用量を削減した．実験では大規模な SIFT 特徴と GIST 特徴のデータセットを用いて精度，処理時間，メモリ使用量の観点で比較を行い，最新の既存手法と比べて高速に最近傍点を得られることを確認した．

キーワード　近似最近傍探索，多次元ハッシング，距離推定，プロダクト量子化，多段階量子化

1. ま え が き

近似最近傍探索とはデータベース $\boldsymbol{X} \in R^d$ に対して，入力ベクトル（クエリ）$\boldsymbol{q}$ に最も距離 $D(\boldsymbol{q}, \boldsymbol{x}) = ||\boldsymbol{q} - \boldsymbol{x}||$ が近いベクトル（最近傍点）$\boldsymbol{x}$ を探索する最近傍探索問題において，探索誤りを許容することで計算時間，メモリ使用量を大幅に削減するものである．最近傍点はクエリと各ベクトルの距離計算によって求めることができる．距離尺度 D は用途によって様々であるが，本稿では特にユークリッド距離を用いる場合に焦点を当てる．D^2 は二乗ユークリッド距離を表すものとする．

近似最近傍探索は高速に最近傍点を探索するためにクエリ入力前にデータ空間をインデクシングし，クエリ入力後に（1）探索領域の絞り込み，（2）限定された探索空間内で距離計算を行う．上述のように，最近傍探索に近似を導入する点には高速化と省メモリ化の２つがあり，（1）と（2）のそれぞれの工程に近似を加えることで実現される．（1）の工程では探索領域を大きく絞り込むことで距離計算の回数を減らして処理を高速化し，（2）の工程では距離計算に必要なデータベース中のベクトルを量子化表現して用いることでメモリ使用量を減らす．

（1）の工程では，予めインデクシングされた領域の中から，最近傍点が含まれる可能性の高い領域をクエリ毎に選択し，これを探索領域とする．以降，探索領域に含まれる点を最近傍候補と呼ぶ．近似最近傍探索における処理時間は，（1）の最近傍候補の抽出時間と（2）の距離計算時間からなり，基本的には（2）の割合が大きい．従って，処理時間を削減するには最近傍候補数を小さくすれば良いが，同時に真の最近傍点が漏れて探索に誤りが生じる可能性が高くなる．また，一般に絞り込みの精度を上げようとすると，最近傍候補の抽出時間が大きくなるため，これを度外視することもできない．

（2）の工程では（1）で選ばれた最近傍候補に対して距離計算を行う．この時に用いられるデータベース中のベクトルを量子化表現して保持することでメモリ使用量を削減できる．一般に，大きな量子化誤差を許容するほどメモリ使用量は減るが，量子化誤差が大き過ぎると，真の最近傍点が最近傍候補に含まれていても，距離計算の誤差によって最近傍点であると判断されずに探索が失敗する．

我々は前報 [1], [2] で Bucket Distance Hashing (BDH) を提案した．これは特にデータベースの大規模に対するスケーラビリティの向上を目的として考案された手法である．これらの文献内ではハッシュを用いた代表的手法である Locality Sensitive Hashing [3] や Spectral Hashing [4]，木構造を用いた代表的手法である ANN [5] と比べて，高速に解を得られることを報告した．しかし，この手法には距離推定にデータ空間の一様分布を仮定していることや，データ空間の次元数の増加に対して推定距離の精度が大きく低下することなどの問題がある．また，大規模データベースを扱うに当たって必須となる，（2）の工程のメモリ使用量の削減に関する議論が行われていない．

そこで，本稿では上述の手法の空間インデクシングにプロダクト量子化を導入し，分布の一般化とデータ空間の高次元化への対応を行い，より一般的なデータに対して高速な探索を可能にする．その後，メモリ使用量削減の手法を導入し，メモリ使用量を制限した時の提案手法の挙動を検証する．実験では，探索精度・処理時間・メモリ使用量の３つの評価指標に基いて，大規模なデータベースを対象とした実験を行い，最新の既存手法と性能比較を行った．画像特徴量としてよく用いられる SIFT 特徴 [6](BIGANN [7]) と GIST 特徴 [8](80 million tiny images [9]) を用いて実験をした結果，提案手法が最も高速に解を得られることが分かった．

2. 関連研究

本節では初めにハッシュ構造を用いた最新の既存手法である Inverted File with Asymmetric Distance Calculation(IVFADC) [10] とその改良手法である Inveted Multi-Index(IMI) [11] について説明し，次に前報 [1], [2] の手法である BDH について説明する．

2.1 IVFADC と IMI

IVFADC や IMI は k-means 法によってデータ空間を粗く量子化（粗量子化）することで，空間をインデクシングする．この時得られるセントロイドの集合を $\boldsymbol{C}$ とし，その数を $|\boldsymbol{C}| = G$ とする．この時，クエリから各領域に属する点までの距離の期待値（推定距離）はその領域のセントロイドまでの距離である．従って，クエリに近いセントロイドを求め，その領域に属する点を最近傍候補とすれば効率がよい．

a） IVFADC の最近傍候補選択

IVFADC は粗量子化に単純なベクトル量子化を用いる．最近傍候補の選択精度を向上させるには空間を細かく分割する（G を大きくする）必要がある．しかし，最近傍候補選択の計算コストは $O(G)$ となり，大きな値をとることができない．従って，最近傍候補選択の精度が十分に得られないという問題がある．

b） IMI の最近傍候補選択

そこで IVFADV の改良手法として提案されたのが IMI [11] である．粗量子化ではベクトル $\boldsymbol{x}$ を 2 つの部分ベクトル $U^1(\boldsymbol{x}), U^2(\boldsymbol{x})$ に分割するプロダクト量子化を行う．得られた部分セントロイドの集合を $\boldsymbol{C}^1, \boldsymbol{C}^2$，その要素数を $|\boldsymbol{C}^1| = |\boldsymbol{C}^2| = g$ とする．セントロイドの集合は $\boldsymbol{C} = \boldsymbol{C}^1 \times \boldsymbol{C}^2$ で得られ，その数は $G = g^2$ となる．$\boldsymbol{C}^1$ の i 番目と $\boldsymbol{C}^2$ の j 番目の要素を並べてできるセントロイドを $\boldsymbol{c}_{ij} = \{\boldsymbol{c}_i^1, \boldsymbol{c}_j^2\}$ と定義すると，クエリ $\boldsymbol{q}$ と $\boldsymbol{c}_{ij}$ に属する点までの推定二乗距離 $F_{ij}(\boldsymbol{q})$ は，クエリから第 m 部分セントロイドの i 番目の要素までの距離を $f_i^m(\boldsymbol{q}) = D^2(U^m(\boldsymbol{q}), \boldsymbol{c}_i^m)$ として，次式で表される．

$$\begin{aligned} F_{ij}(\boldsymbol{q}) \equiv & D^2\left(\boldsymbol{q}, \boldsymbol{c}_{ij}\right) \\ = & D^2\left(U^1(\boldsymbol{q}), \boldsymbol{c}_i^1\right) + D^2\left(U^2(\boldsymbol{q}), \boldsymbol{c}_j^2\right) \\ = & f_i^1(\boldsymbol{q}) + f_j^2(\boldsymbol{q}) \end{aligned} \tag{1}$$

従って，セントロイドをクエリに近い順に選択する問題は，2 つの部分距離リスト $\{f_i^1\}, \{f_j^2\}$ の中から 1 つずつ選択して，その和 F_{ij} が小さくなる i と j の組み合わせを探索する問題に帰着する．IMI ではこの問題を Multi-Sequence アルゴリズムと呼ばれる組み合わせ探索法を用いて解く．このアルゴリズムは，ある時点で最も距離が小さい添え字の組み合わせを生成する毎に，次に小さい組み合わせを生成する可能性がある添え字の組を添え字候補を追加し，その中から次の組み合わせを選択する処理を繰り返す．そして，得られた最近傍候補数が L 点に達した時点で探索を終了する．

空間の分割数 G が等しい時，プロダクト量子化はベクトル量子化に比べて精度は悪くなる．しかし，プロダクト量子化を用いることで Multi-Sequence アルゴリズムが適用でき，計算コストが $O(g \log g) = O(G^{1/2} \log G)$ で最近傍候補を得られるため，探索の高速化が実現できる．文献 [11] では，IVFADC に比べて高速に解を得られることが報告されている．

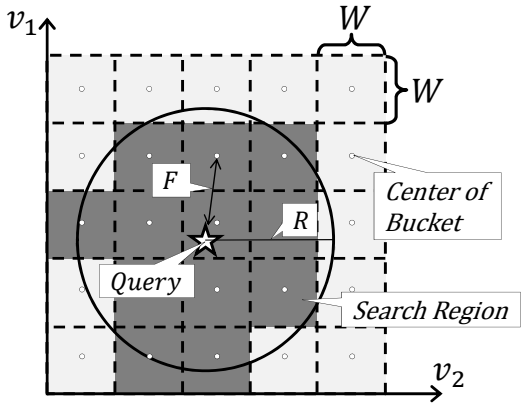


図 1 Space indexing of BDH

c） 量子化表現を用いた距離計算

IVFADC や IMI では，メモリ使用量削減の為に，データベース中のベクトルをプロダクト量子化してそのインデックスを登録する．量子化の際，ベクトルそのものを量子化するのではなく，各データ $\boldsymbol{x}$ が属する領域のセントロイドを $\boldsymbol{c}(\boldsymbol{x})$ として，そこからの残差ベクトル $r(\boldsymbol{x}) = \boldsymbol{x} - \boldsymbol{c}(\boldsymbol{x})$ を量子化する．ここで，クエリを $\boldsymbol{q}$，あるベクトルを $\boldsymbol{x}$，量子化器を Q とすると，得られる D の近似値 $\ddot{D}$ は次式で表される．

$$\ddot{D}(\boldsymbol{q}, \boldsymbol{x}) = D(\boldsymbol{q} - \boldsymbol{C}(\boldsymbol{x}), Q(r(\boldsymbol{x}))) \tag{2}$$

残差ベクトルはベクトルそのものに比べて分散が小さく，$Q(\boldsymbol{x})$ と $Q(r(\boldsymbol{x}))$ では，$Q(r(\boldsymbol{x}))$ の方が同じメモリ使用量でも量子化誤差が減少する．

2.2 Bucket Distance Hashing

BDH は多次元ハッシュを用いて空間インデクシングを行うことで，区分けされた領域毎に含まれる点とクエリの距離を推定し，得られた推定距離を基に最近傍候補を選択する手法である．

2.2.1 空間インデクシングと推定距離

$\boldsymbol{V}$ を任意の d 次元の正規直交基底とする．BDH では $\boldsymbol{V}$ のうち M 個の基底に着目し，図 1 のように一辺が W の格子状の矩形領域に等分割する．ただし，図 1 は $M = 2$ とした場合の様子である．分割された各領域をバケットと呼び，各バケットは多次元ハッシュによって表現される．$\boldsymbol{x}$ を任意の点，$\boldsymbol{v}^m$ を $\boldsymbol{V}$ の第 m 基底として，ハッシュ関数 $H(\cdot)$ は次のようになる．

$$H(\boldsymbol{x}) = \{h^1(\boldsymbol{x}), h^2(\boldsymbol{x}), \ldots, h^M(\boldsymbol{x})\} \tag{3}$$

$$h^m(\boldsymbol{x}) = \left\lfloor \frac{\boldsymbol{v}^m \boldsymbol{x}}{W} \right\rfloor \tag{4}$$

つまり，バケットは $H(\boldsymbol{x})$ が等しくなるような $\boldsymbol{x}$ の集合である．この時バケット数 G は，データが存在する範囲を覆うのに十分なハッシュ値の個数を g^m とすることで，次式のようになる．

$$G = \prod_m g^m \tag{5}$$

$$g^m = \max h^m(\boldsymbol{x}) - \min h^m(\boldsymbol{x}) + 1 \tag{6}$$

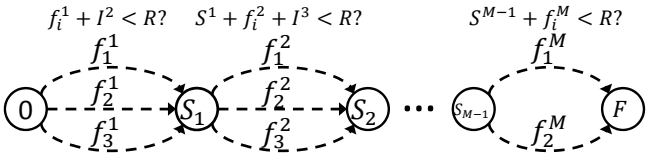


図 2　Reference bucket selection with graph path finding

H をハッシュ値リストとして，IMI のセントロイド $\boldsymbol{c}_H$ を一般化すると，

$$\boldsymbol{c}_H = \left\{ \left(h^1 + \tfrac{1}{2}\right) W, \ldots, \left(h^M + \tfrac{1}{2}\right) W \right\} \tag{7}$$

となる．$(i+\frac{1}{2})W$ はハッシュ値 $h^m = i$ となるようなバケットの中心座標を示している．上式から，推定二乗距離（バケット距離）F_H は次式で表される．

$$\begin{aligned} F_H =& D^2(\boldsymbol{q}, \boldsymbol{c}_H) \\ =& \sum_m D^2 \left(\boldsymbol{v}^m \boldsymbol{q}, (h^m + \tfrac{1}{2}) W\right) \end{aligned} \tag{8}$$

これにより，図 1 のように探索領域をクエリを中心とする近似的な超球領域に絞り込むことができる．上記の直交基底 $\boldsymbol{V}$ は任意のものでよいが，距離情報を M 次元に縮退させて扱うため，主成分分析（PCA）によって得られた上位の主成分基底を用いるのが効率的である場合が多い．

2.2.2　高速な近傍バケット探索

近傍バケットの探索には IMI と同様に，部分距離リスト $\{f_i^m\}$ を生成し，その和 F_H が小さくなる組み合わせを探索する問題に帰着させる．f_i^m を次式で定義する．

$$f_i^m = D^2(\boldsymbol{v}^m \boldsymbol{q}, (i + \tfrac{1}{2}) W) \tag{9}$$

IMI では組み合わせ探索問題を Multi-Sequence アルゴリズムで解いた．BDH では，組み合わせを生成する部分距離リスト $\{f_i^m\}$ の数 M は数十に及び，添字候補の数が組み合わせ爆発を起こし，効率的な探索を行うことはできない．例えば，W を大きく取って全ての m に対して $g^m = 2$ となった場合，F が最も小さくなる組み合わせはクエリが入ったバケットを示す $H(\boldsymbol{q})$ であるが，次の候補は残り全ての組み合わせ $2^M - 1$ 通りある．

そこで，我々はこの組み合わせ問題を経路探索問題に変換し，効率的な経路の枝狩りアルゴリズムを導入することで高速に解を得る方法を取った．BDH では，距離が小さいバケットを順に参照するのではなく，距離が閾値 R 以下となる領域を，その順序を考慮せずに探索することで，問題を簡略化する．これにより，組み合わせリストの数が大きくなっても，高速に問題を解くことができる．以下，距離が R 以下となるバケットを高速に見つけるためのアルゴリズムについて述べる．

我々はこの組み合わせ問題を図 2 で示す経路探索問題として扱う．左端の根ノードから右端の終端ノードに至るまでの間に M 回エッジを選択し，エッジに付与された重み f_i^m を得る．これは m 番目に選択したエッジのインデックスがハッシュ値 h^m に対応しており，終端ノードでの重みがバケット距離 $F_H(\boldsymbol{q})$ となる．従って，バケット距離が R 以下となるバケットを探す問題は図 2 の “重み付き有向グラフにおける重みの和が R 以下” となる経路を探索する問題と見なせる．この経路探索問題は前処理を加えることで大部分の経路選択を枝刈りすることができ，高速に解くことができる．以下に枝狩りアルゴリズムを示す．

ここで，S^m を根ノードからの重みの和とし，I^m を m 番目以降の重みの最小和とする．

$$I^m = \sum_{l=m}^{M} \min_i f_i^l \tag{10}$$

m 回のエッジを選択した後の重みの和 S^m について考えると，もし $S^m + I^{m+1} > R$ であれば，この先どのような経路を選択しても必ず F は R を超えることが分かり，その経路を枝狩りすることができる．従って m 番目のエッジを選択する時，次式を満たす重み f_i^m を持つエッジのみを選択する．

$$S^{m-1} + f_i^m + I^{m+1} < R \tag{11}$$

このような経路の枝刈り条件を加えることで無駄な経路探索を排除し，$F_H < R$ を満たす組み合わせのみを探索することができる．この処理の計算コストは参照するバケット数を n としてオーダー $O(n)$ であり，データサイズやバケット数（空間の分割数）に依存せず，高速に参照バケットを特定することができる．

3. 提 案 手 法

提案手法は前報の BDH [1], [2] を発展させたものであり，空間インデクシングのためのハッシュ関数を改良することで，BDH の問題であるデータの一様分布の仮定を無くし，データ空間が高次元化した場合の性能の低下を緩和する．

3.1　空間インデクシング

前報 [1], [2] ではデータ空間の一様性を仮定し，データ空間を格子状に等分割した．これは，選択した基底によって張られる M 次元空間をスカラ量子化によって粗量子化したことに等しい．しかし，一般に実データは一様でないため，データの分布を考慮しない量子化は最適とは言えない．また，実データにおいては基底が直交していてもそれらが互いに独立であるとは言えず，スカラ量子化では量子化効率が悪い．

バケット数 G は式 (5) のように M に対して指数的に大きくなる．バケットの数が大きくなれば，精度は上がるが隣接バケットを探索する処理に時間がかかるようになる．従って，安易に大きな M を用いることはできず，ベクトルが高次元であった場合には全次元数に対してインデクシングに使う M が相対的に小さくなり，距離推定の精度が得られずに近似最近傍探索全体の性能の低下を招く．経験的にバケット数がデータセットサイズ $|\boldsymbol{X}|$ に等しくなる程度が最もバランスが良いことが分かっている．

そこで，提案手法では直交基底 $\boldsymbol{V}$ の中から，P 本の直交基底から成るセットを M 組選択し，データ空間を M 個の P 次元部分空間の直積で表現する．ここで，$\boldsymbol{V}^m$ を m 番目の部分空間を張る P 個の直交基底とする．インデクシングの際には

部分空間毎に k-means アルゴリズムを用いてセントロイドの集合 $\boldsymbol{C}^m$ を求め，量子化誤差の最小化を図る．これは上記の基底によって張られた PM 次元空間に射影されたベクトルを，M 個の P 次元部分ベクトルに分けてプロダクト量子化することに等しい．$\boldsymbol{C}^m$ の i 番目の部分セントロイドを $\boldsymbol{c}_i^m$ と表せば，式 (4) のハッシュ関数は次式と置換される．

$$h^m(\boldsymbol{x}) = \arg\min_i D(\boldsymbol{c}_i^m, \boldsymbol{V}^m\boldsymbol{x}) \tag{12}$$

3.2 バケット距離

ここでは推定誤差が最小となるような距離の推定量を導き，バケット距離を定義する．各基底間に相関がないと仮定（主成分基底を用いれば2次までの無相関が保障される）すれば，各基底方向の推定距離の誤差を独立に最小化すればよいので，1次元の二乗距離の誤差を最小化することを考える．

最小化問題を次のように定義する．データ x は確率密度関数 $P(x)$ に従うものとし，x がある領域 z に存在する事象を Z とすれば，領域 z 内にあるデータの分布は $P(x|Z)$ と表される．この時，$\tilde{x} = \{x \in z\}$ と置くことで $P(\tilde{x}) = P(x|Z)$ となる．このような条件の下でクエリ q と $\tilde{x}$ の二乗距離 $D^2(q,\tilde{x})$ の推定量を e とすると，誤差の最小化問題は次のようにおける．

$$\min \mathrm{E}_{\tilde{x}}[\{D^2(q,\tilde{x}) - e\}^2] \tag{13}$$

この式の形は分散の定義式と同じであるため，式 (13) を最小化する e は以下のように $D^2(q,\tilde{x})$ の期待値として得られる．

$$\begin{aligned} e =& \mathrm{E}_{\tilde{x}}[D^2(q,\tilde{x})] \\ =& q^2 - 2\mathrm{E}[\tilde{x}]\,q + \mathrm{E}[\tilde{x}^2] \\ =& (q - \mathrm{E}[\tilde{x}])^2 + \mathrm{Var}[\tilde{x}] \end{aligned} \tag{14}$$

ここで得られた推定量は領域の重心との距離ではないことに注意する．以上の結果から，ハッシュ値リストが H であるようなバケット距離 F_H を次のように定義する．

$$F_H = \sum_{m=1}^{M} f_{h^m}^m \tag{15}$$

$$f_{h^m}^m = D^2(\boldsymbol{q}, \boldsymbol{c}_{h^m}^m) + \sum \mathrm{Var}[\tilde{x}_p^m] \tag{16}$$

$$\tilde{x}_p^m = \{\boldsymbol{u}_p^m\boldsymbol{x} | h^m(\boldsymbol{x}) = i^m\} \tag{17}$$

3.3 部分セントロイドの数

次に各部分空間に与えるセントロイドの数 g^m を考える．一般的なプロダクト量子化では，g^m は各部分空間で共通の値を用いるが，PCA を用いた場合など，各部分空間のデータの広がりに偏りがある場合は量子化効率が悪く，量子化誤差を最小化するには g^m を各部分空間の広がりに合わせて調節する必要がある．

式 (13) の誤差を式 (14) の推定量 e を用いて計算し直すと，次のようになる．

$$\begin{aligned} &\mathrm{E}_{\tilde{x},q}[\{D^2(q,\tilde{x}) - e)\}^2] \\ =& 4\mathrm{E}[q^2](\mathrm{E}[\tilde{x}^2] - \mathrm{E}[\tilde{x}]^2) + \mathrm{E}[\tilde{x}^4] - \mathrm{E}[\tilde{x}^2]^2 \\ &+ 4\mathrm{E}[q](\mathrm{E}[\tilde{x}]\mathrm{E}[\tilde{x}^2] - \mathrm{E}[\tilde{x}^3]) \\ =& 4\mathrm{E}[q^2]\mathrm{Var}[\tilde{x}] + \mathrm{Var}[\tilde{x}^2] + 4\mathrm{E}[q](\mathrm{E}[\tilde{x}]\mathrm{E}[\tilde{x}^2] - \mathrm{E}[\tilde{x}^3]) \\ \simeq& 4\mathrm{E}[q^2]\mathrm{Var}[\tilde{x}] \end{aligned} \tag{18}$$

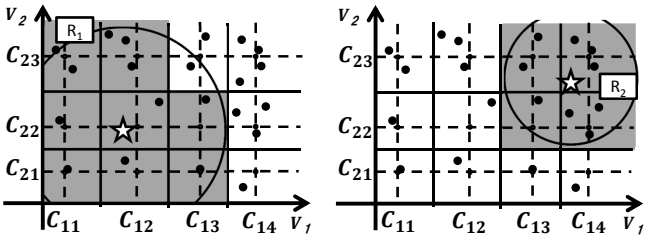


図 3　クエリ適用探索

最後の近似は $\tilde{x}$ は領域 z 内に縛られているために q に対して分散が小さく，$\mathrm{E}[q^2]$ が他の要素に比べて非常に大きくなるため為された．従って空間インデクシングの際には，分割された領域内の分散 $\mathrm{Var}[\tilde{x}]$ に注意すればよく，これは各部分空間の量子化誤差に相当する．以上から，BDH では各部分空間の量子化誤差の最大値が最も小さくなるように，分散が大きい基底セットほど g^m を大きく設定する．

3.4 クエリ適応探索

前報 [2] では予めパラメータとして与えられた閾値 R を基に，バケット距離が小さいものを最近傍候補としたが，文献 [12] で述べられているように，探索半径 R をパラメータとして与えると，データが疎な領域では探索領域が狭過ぎて精度が得られず，その一方で密な領域では探索領域が広過ぎて距離計算が過剰となる．

そこで我々は，図 3 のようにクエリ周辺のデータ密度の差に応じて探索領域の大きさを変化させるため，R を δ ずつ大きくして，最近傍候補数が L を超えた所で探索を終了する．その為，式 (11) の経路選択の枝刈りの条件を置き換える．m 番目以降の最大重み和を

$$J^m = \sum_{l=m}^{M} \max_i f_i^l \tag{19}$$

$t = 1, 2, 3\ldots$ とすると，次式を満たすエッジのみを選択する．

$$\begin{cases} \delta t \leqq S^{m-1} + f_i^m + J^{m+1} \\ S^{m-1} + f_i^m + I^{m+1} < \delta(t+1) \end{cases} \tag{20}$$

式 (20) の条件によって得られるバケットはクエリを中心とする半径 $\delta(t+1)$ の球から半径 δt の球を取り除いた領域に存在するものである．

3.5 多段ベクトル量子化

ここではまず IVFADC [10] と同様の方法でメモリ使用量の削減を図る．この時，復元されたベクトルの精度は1段目の空間インデクシングの量子化と，2段目のベクトル保存のための量子化の精度で決まる．

1段目の量子化について考えると，セントロイド数 G が共通ならば，図 4 に示すように M が比較的大きい BDH は一般に量子化誤差が大きくなる．また1段目の量子化誤差が等しい場

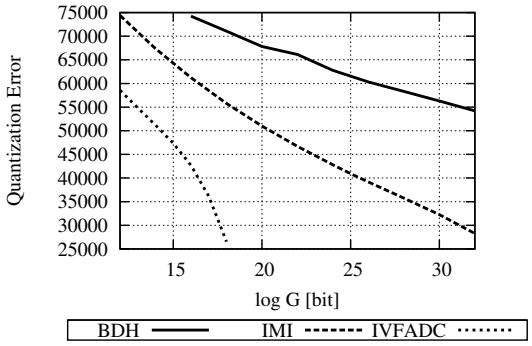


図 4　領域分割数 G と量子化誤差の関係

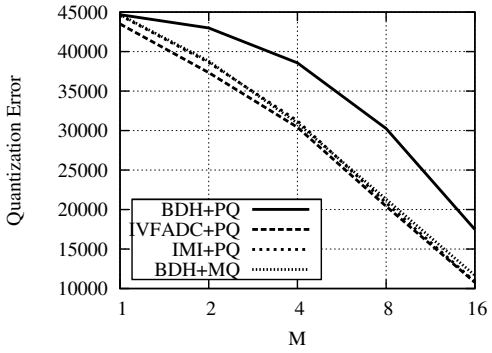


図 5　ベクトル分割数 M と量子化誤差の関係

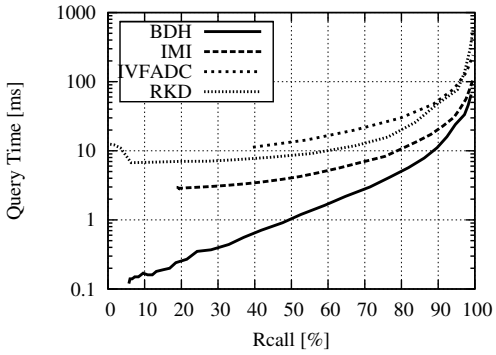


図 6　処理時間と Recall の関係　（SIFT 1 億点）

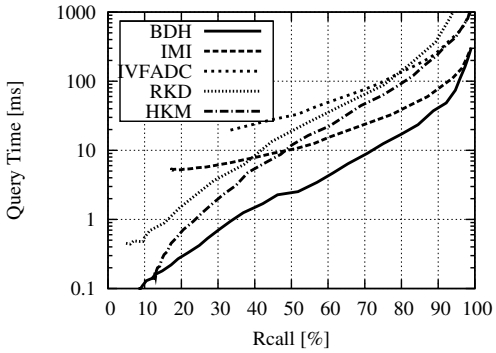


図 7　処理時間と Recall の関係　（GIST 1000 万点）

合でも，残差ベクトル $r(\boldsymbol{x})$ をプロダクト量子化した時の量子化誤差は提案手法 (BDH+PQ) が最も大きくなった (図 5)．これは 3.3 節で述べたように，主成分基底を用いたことで BDH の残差ベクトル $r(\boldsymbol{x})$ の分散に偏りが生じ，量子化効率が悪くなる為である．

ベクトル量子化すれば分散の偏りは量子化器に吸収されるが，単純なベクトル量子化ではセントロイドと同じ数だけコードブックを保持しておく必要があり，その分のメモリ使用量が大きくなり過ぎるのために実用的ではない．そこで，我々は残差ベクトルの量子化に多段ベクトル量子化 [13] を導入する．

多段ベクトル量子化とは，セントロイドとの残差ベクトルを M^* 段階に分け再帰的にベクトル量子化するものであり，コードブックに必要なメモリ使用量を $1/M^*$ 乗に抑えることができる．この結果を図 5 の BDH+MQ として示す．図より，M^* の増加に伴って量子化誤差が IVF+PQ と IMI+PQ のような既存手法と同様に減少していることがわかる．従って，提案手法にとってはプロダクト量子化によるメモリ削減ではなく，ベクトル量子化を再帰的に適用する方法の方が適当と考えられる．実際に実験したところ，この考察を支持する結果が得られた．

4. 実　　験

本節では初めに，メモリ使用量を制限せずに精度と処理時間の関係を検証し，次にメモリ削減を適用して，使用できるメモリ使用量に上限を設けた場合の精度と処理時間の関係を検証した．問題設定はクエリに近い K 点を探索する K 近傍探索問題とした．

表 1　使用パラメーター一覧

手法：パラメータ	SIFT 1 億点	GIST 1000 万点
BDH : $\log_2 G$, P	28 , 5	24 , 5
Multi-Index : $\log_2 G$	20	22
IVFADC : $\log_2 G$	14	14
Rand kd-tree : 木の数	8	16
階層的 k-means : 分岐数		32

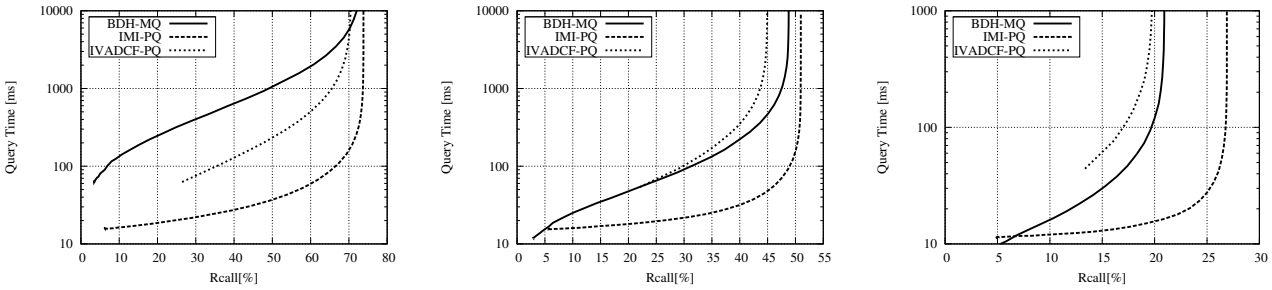
4.1 実 験 条 件

実験には BIGANN データセット [7] の SIFT 特徴量と 80 million tiny images [9] の GIST 特徴量を用いた．クエリはともに 1000 個とし，探索精度は平均再現率，処理時間はクエリベクトル 1 個当たりの処理時間の平均である．SIFT の学習にはデータセットの初めの 100 万点，GIST の学習には 10 万点を用いた．実験に用いた CPU は Opteron(tm)6174(2.2GHz) であり，実験は全てシングルコアで行った．

4.2 処理時間と精度の関係

本節では提案手法と各比較手法の処理時間と精度の関係を示す．$K = 1$ とした．ここではメモリ削減をしないため，どの手法も探索領域を広げることで必ず 100% の精度を達成できる．実験には SIFT が 1 億点，GIST が 1000 万点のデータセットを用いた．比較手法は木構造を用いた手法として Randamized kd-tree [14]，階層的 k-means [15]，ハッシュ法を用いたものとして IVFADC，IMI を用いた．結果を図 6，7 に，用いたパラメータを表 1 に示す．図の横軸は処理時間，縦軸は精度を表している．

提案手法はいずれの特徴量で比較しても，常に既存手法に比べて高速に解を得られることが確認できた．精度が 90% を超



(a) 1/4(31.7GB) に圧縮　(b) 1/8(15.8GB) に圧縮　(c) 1/16(7.9GB) に圧縮

図 8　メモリ使用量に上限を設けた場合の処理時間と Recall の関係

えるような高精度帯においては各手法間にそれほど大きな違いは見られないが，50% 以下の低精度帯においては提案手法が明らかに他と比べて高速である．特に点数の多い図 6 で顕著である．

この原因は最近候補選択のオーバーヘッドである．IVFADC は近傍セントロイドの探索に G 回の距離計算を行い，IMI は部分距離リスト $\{f_i^m\}$ の生成に $2G^{1/2}$ 回の距離計算とそのソートを必要とする為，低精度であっても処理時間が小さくならない．しかし，空間インデクシングの量子化精度は BDH に比べて高いので，精度に対するグラフの上昇は緩やかになる．

4.3　量子化によるメモリ削減

10 億点の SIFT 特徴量に対して $K = 1000$ 近傍を解として，量子化によるメモリ削減を施した場合の処理時間と精度の関係を調査した．データ構造を用いずに線形探索を行った場合のメモリ使用量は約 127GB であった．そこで，メモリ使用量の上限を上記の 1/4(31.7GB)，1/8(15.8GB)，1/16(7.9GB) のように変化させた時の結果を図 8(a)〜8(c) に示す．

実験の結果同一精度で比較すると，メモリを削減した場合は IMI が最も高速であることが分かった．この最も大きな原因は 1 段目の量子化誤差の差である．今回の実験ではメモリ使用量の上限を守るため，分割数は最大でも 1/4,1/8 で $G = 2^{26}$,1/16 で $G = 2^{25}$ である．ところが，表 1 を見ると BDH は約 30[bit] 程度が最適であると予想できる．その一方で IMI は約 24[bit] 程度と考えられるため，まだ余裕がある．従って，ハッシュに要するメモリ使用量の上限の問題から，BDH は速度に関して最適なパラメータを選ぶことができず，IMI に劣る結果になったものと考えられる．

5. ま と め

前報 [1], [2] を改良し，より一般的なデータ分布に適用した探索法を提案した．実験の結果，処理速度のみで比較すると提案手法が最も高速であることがわかった．しかし，メモリ使用量に制限を設けると，ハッシュテーブルに必要なメモリ使用量の問題から，提案手法は十分な距離計算精度を得ることができなかった．

今後の課題は 1 段階目のプロダクト量子化における部分空間の数 M を大きく維持したまま，量子化誤差を小さく抑えることができるようなインデクシングを考案し，メモリ使用量を減らした時の精度向上を目指すことである．

謝辞　本研究の一部は科学技術戦略推進費「安全・安心な社会のための犯罪・テロ対策技術等を実用化するプログラム」の一環で実施され，科研費補助金基盤研究 (B)(22300062) ならびに科学技術振興機構 CREST の補助を受けた．ここに記して感謝する．

文　献

[1] 佐藤智一，武藤大志，岩村雅一，黄瀬浩一，“バケット距離に基づく近似最近傍探索，” DEIM2011，pp.1–6，Feb. 2011.
[2] 佐藤智一，岩村雅一，黄瀬浩一，“概算距離の精度向上による近似最近傍探索の高速化，” 信学技報.PRMU111-194，pp.61–66，2011.
[3] P. Indyk and R. Motwani, “Approximate nearest neighbors: towards removing the curse of dimensionality,” Proc 30th Symposium on Theory of Computing, pp.604–613, 1998.
[4] Y. Weiss, A. Torralba, and R. Fergus, “Spectral hashing,” Proc. NIPS2008, pp.1753–1760, 2008.
[5] S. Arya, D.M. Mount, N.S. Netanyahu, R. Silverman, and A.Y. Wu, “An optimal algorithm for approximate nearest neighbor searching in fixed dimensions,” Journal of the ACM, vol.45, no.6, pp.891–923, Nov. 1998.
[6] D.G. Lowe, “Distinctive image features from scale-invariant keypoints,” IJCV, pp.91–110, 2004.
[7] “Datasets for approximate nearest neighbor search,” http://corpus-texmex.irisa.fr/.
[8] A. Oliva and A. Torralba, “Modeling the shape of the scene: A holistic representation of the spatial envelope,” IJCV, vol.42, no.3, pp.145–175, 2001.
[9] “Tiny Images Dataset,” http://groups.csail.mit.edu/vision/TinyImages/.
[10] H. Jégou, M. Douze, and C. Schmid, “Product quantization for nearest neighbor search,” IEEE Trans. TPAMI, vol.33, no.1, pp.117–128, 2011.
[11] A. Babenko and V. Lempitsky, “The inverted multi-index,” Proc. CVPR2012, pp.3069–3076, 2012.
[12] W. Dong, Z. Wang, W. Josephson, M. Charikar, and K. Li, “Modeling LSH for performance tuning,” Proc. CIKM2008, pp.669–678, 2008.
[13] A. Gersho and R.M. Gray, Vector quantization and signal compression, vol.159, Springer, 1992.
[14] C. Silpa-Anan and R. Hartley, “Optimised kd-trees for fast image descriptor matching,” Proc. CVPR2008, pp.1–8, 2008.
[15] D. Nister and H. Stewenius, “Scalable recognition with a vocabulary tree,” Proc. CVPR2006, vol.2, pp.2161–2168, 2006.