高次元空間における効率的な近似k Nearest Neighbor検索アルゴリズム

新井　悠介†　　天方　大地†,††　　藤田　澄男†††　　原　　隆浩†

† 大阪大学大学院情報科学研究科　〒 565–0871 大阪府吹田市山田丘 1-5
†† JST さきがけ
††† ヤフー株式会社
E-mail: †{arai.yusuke,amagata.daichi,hara}@ist.osaka-u.ac.jp, ††sufujita@yahoo-corp.jp

あらまし　近似 k Nearest Neighbor（AkNN）検索は，コンピュータビジョンおよび機械学習をはじめとする様々な分野で利用されている．AkNN 検索のアプリケーションでは，高次元データに対して高速かつ高精度に動作するアルゴリズムが必要とされている．そのアプローチとして，グラフインデックスを用いたアルゴリズムが，ハッシュや量子化を用いたアルゴリズムよりも高性能であることが知られている．しかし，既存のグラフインデックスを用いたアルゴリズムは極めてヒューリスティックであるため，その性能の理論的根拠はない．本稿では，Locality Sensitive Hashing と近接グラフを用いた新しい AkNN 検索アルゴリズムを提案する．提案アルゴリズムの性能は理論的に裏付けされている．実世界のデータを用いた実験により，提案アルゴリズムは既存の state-of-the-art と比較して高速かつ高精度に動作することを示した．

キーワード　kNN 検索，高次元データ

1　はじめに

k Nearest Neighbor（kNN）検索は，コンピュータビジョン [18], [19]，機械学習 [29]，マルチメディア検索 [33]，およびデータベース [26] など，様々な分野で利用されている操作である．そのため，これまでに多くの効率的な kNN 検索アルゴリズムが提案されてきた．例えば kd 木 [9] をはじめとする木構造を用いたインデックスを利用するアルゴリズムが知られている．これらのアルゴリズムにより，低次元のデータ集合に対して厳密な kNN を高速に検索できる．しかし近年，高次元データを扱うアプリケーションが増えてきた [5], [28]．次元の呪いと呼ばれる現象により，高次元空間では，木によるインデックスを利用しても効率的な kNN 検索ができない [34]．そこで，近似 kNN（AkNN）検索アルゴリズムが注目を集めている．多くのアプリケーションは，わずかな誤差は許容して，高速なアルゴリズムを求めている．

AkNN 検索の代表的なアプローチは，ハッシュ [11]，量子化 [22]，および近接グラフ [18] である．ハッシュによるアプローチでは，Locality Sensitive Hashing（LSH）と呼ばれるハッシュ関数を利用する．LSH は，距離の近いデータ同士が衝突する確率が高いという性質を持ったハッシュ関数である．量子化は，ベクトルを分割し，それぞれを既知の近傍のベクトルで近似する．検索の際は，転置ファイルを利用して AkNN を計算する．ハッシュおよび量子化による方法は，どちらもデータの次元削減を行う手法である．近接グラフによるアプローチでは，各データがノードに相当し，データ同士の距離が小さい時にエッジを作る．そして，クエリが与えられると，近接グラフの始点から始めて，最もクエリに接近するエッジを貪欲に辿ることにより，AkNN を求める．

これらの 3 つのアプローチのうち，近接グラフをインデックスとして用いる方法が，最も高精度かつ高速である [15], [26], [29]．しかし，現在最高性能とされているグラフによる検索アルゴリズムは，極めてヒューリスティックであるため，その性能の理論的解析は不可能である．すなわち，そのアプリケーションは，ハイパーパラメータをどのように変更すれば精度と速度のトレードオフを適切に調節できるのかを把握できない．実際, [15], [29] などは精度と速度のトレードオフを直感的に調節するパラメータを持たない．さらに，これらのアルゴリズムでは，階層構造 [29] および単調経路 [3], [15], [18] などの複雑なグラフ構造を導入しているため，その解析と実装が難しい．そして，これらのグラフ構造は，少ない距離計算回数でクエリに接近できることを理論的に保証できていない．

これらの問題点を解消するため，我々はハッシュと近接グラフの両方を利用した新しい AkNN 検索アルゴリズムを提案する．本稿の貢献を以下に示す．

- ハッシュテーブルと近接グラフを統合した新しいデータ構造を提案する．このデータ構造はシンプルであるだけでなく，既存の AkNN 検索アルゴリズムと同じ時間計算量で**厳密な** kNN を求められる．
- 上で述べたデータ構造は，理論的には有効であるものの，前処理の計算コストとメモリ消費量が大きい．そこで我々はより実践的なアルゴリズムである LGTM を導入する．LGTM は高速かつ高精度に AkNN を計算でき，その空間計算量は $O(n)$ である．ただし，n は与えられたデータ集合の要素数である．そして，精度と速度のトレードオフは一つのパラメータにより調整可能である．このパラメータが大きくなると，LGTM はより高精度な AkNN を返却することが理論的に保証されている．さらに，既存のアルゴリズムと異なり，LGTM はマルチスレッディングに対応している．これにより，そのパラメータ

が大きくなっても効率的に実行できる．

- 3 つの実世界のデータセット（SIFT，GIST，および Deep）を用いた実験を行う．実験により，LGTM は既存の最高性能の AkNN 検索アルゴリズムよりも高速かつ高精度に検索できることを確認する．

2 関 連 研 究

木構造を用いる方法． R 木 [8] および kd 木 [9] などの木構造は kNN 検索のためのインデックスとして最も代表的なデータ構造である．木を用いたアプローチでは，データ空間を再帰的に分割し，その分割領域を階層的に管理する．これにより，検索の際に，全ての領域をスキャンするのではなく，一部の分割領域に含まれるデータとの距離を計算するだけで厳密な kNN を計算できる．しかし，木を用いたアルゴリズムは，次元の呪いにより，高次元空間では性能が低下する．

ハッシュを用いる方法． 近似により kNN 検索を高速化するため，ハッシュを用いた AkNN 検索アルゴリズム [23], [24], [32] がこれまでに数多く提案されてきた．一般的に，ハッシュによる方法では，データ同士の距離が小さい時に衝突する可能性が高いハッシュ関数を設計し，これを用いて元の高次元データをより低次元の空間に写像する．

これを実現する方法の一つが Locality Sensitive Hashing (LSH) [1], [2], [11] である．LSH の特長は，ハッシュ関数を適切に設計することにより，AkNN 検索の精度を理論的に保証できることである．我々は，LSH のこの性質を提案手法に利用している（4. 2 節）．

LSH がデータの分布に依存しないハッシュ関数であるのに対し，機械学習によりデータの分布を推定して精度を向上するアプローチ [10], [20], [27] も盛んに研究されている．しかし，この方法はラベルのアノテーションや学習などの前処理の計算コストが高い．例えば, [19], [33] では，GPU を使用しても前処理に数時間かかる．さらに，学習による方法では理論的な保証が失われるため，我々はこの方法は使わない．

量子化を用いる方法． 量子化による方法では，与えられたベクトルを，コードブックの中で最も距離の小さいベクトルに割り当てる．そして，ハッシュと同様に，転置インデックスを使用する．クエリが発行されると，クエリをコードブックの中で最も距離の小さいベクトルに割り当て，同じベクトルに割り当てられたベクトルの中で kNN を検索する [16], [17], [22]．また，その量子化誤差を減らすための方法 [6], [31], [35] および転置インデックスによる検索アルゴリズムの性能改善 [4], [21], [25] についても数多く提案されている．量子化による方法の大きな特長は，メモリ消費量が小さいことである．しかし，AkNN 検索の精度を向上するためには，量子化ビット数を大きくしなければならない．これにより，メモリ消費量は大きくなり，検索速度も低下する．

近接グラフを用いる方法． 近接グラフは，ノード（データ）同士の距離が小さい時にエッジが存在するデータ構造である．近接グラフを用いた AkNN 検索アルゴリズムは，クエリが与えられた時，あるノードの隣接ノードを辿るとクエリに近付くという直感に基づいている．

Efanna [14] はグラフインデックスを辿って AkNN 検索を行うアルゴリズムである．ある近接グラフが与えられた時，Efanna は木構造のインデックスを用いてクエリに近いノードを見つける．そして，そのノードを始点として隣接ノードにアクセスし，AkNN を得る．しかし，Efanna の始点の見つけ方は効率的ではなく，理論的な根拠もない．すなわち，始点はクエリから遠くなることもあり得るため，高速に AkNN を検索できない場合がある．我々の提案手法である LGTM も，Efanna と同様にクエリに近い始点を探してからグラフ上の検索を行うが，LGTM は Efanna よりも理論的根拠に基づく方法を採用している．

近年の研究 [14], [18], [29], [30] では，AkNN 検索のために，アルゴリズム 1 に示す貪欲法を使用しており，このアルゴリズムに適したグラフ構造を提案している．まず，あるノードから目的のノードまでのパスについて考える．このパスに含まれるノードを順番に訪問する中で，目的のノードまでの距離が単調減少する時，そのパスを単調経路という．与えられたグラフの任意の二つのノード間に単調経路が存在する場合，そのグラフは Monotonic Search Graph であるという [12]. Monotonic Search Graph の構築には $\Omega(n^2)$ の時間がかかるため，FANNG [18] および NSG [15] では近似 Monotonic Search Graph を構築している．一方, [30] は Small World Network を提案している．これにより，任意の二つのノード間を $O(\log n)$ ホップで辿れる．HNSW [29] は Small World Network を階層的に配置する．ある層のノード集合は一つ下の層のノード集合の部分集合である．検索の際には，上の階層から始めて下の階層へと下降することで，一層の Small World Network よりも効率的にクエリに接近できる．

NSG と HNSW はハッシュおよび量子化によるアプローチよりも高速かつ高精度に AkNN 検索を実行できる [15], [26], [29]. しかし，NSG には問題点がある．検索の際に必ず単調経路を辿るという保証がないため，局所解に陥り，精度が低下する場合がある．また HNSW は極めてヒューリスティックであるため，理論的に精度を向上できるチューニング方法がない．

3 問 題 定 義

X を d 次元ユークリッド空間 $\mathbb{R}^d$ 上の点の集合とし，$|X| = n$ とする．2 点 $x_i, x_j \in \mathbb{R}^d$ の距離 $dist(x_i, x_j)$ はユークリッド距離とする．本問題である kNN 検索を以下のように定義する．

定義 1（kNN 検索）． 点集合 X，クエリ q，検索結果の数 k が与えられたとする．kNN 検索は，$\forall x \in S$, $\forall x' \in X - S$, $dist(x, q) \leqq dist(x', q)$ である点集合 S を求める操作である．

本稿では d が大きいデータを想定しているため，AkNN 検索に焦点を当てる．また，先行研究 [15], [18], [29] と同様，AkNN 検索の精度は Recall により評価する．AkNN 検索の結果を S' とすると，Recall は $\frac{|S \cap S'|}{k}$ である．

Algorithm 1: GREEDY-AkNN-SEARCH

Input: X, a proximity graph, q, k, ϵ, and a start point s

$X_{visit} \leftarrow s$
$C \leftarrow \langle s, dist(s,q)\rangle$ (C is sorted by $dist(\cdot, q)$)
$Q \leftarrow C$, $\tau \leftarrow \infty$
while $Q \neq \varnothing$ **do**
  $\langle x, \cdot\rangle \leftarrow$ the top of Q
  Pop $\langle x, \cdot\rangle$ from Q
  for *each* $(x, x') \in E(x)$ *s.t.* $x' \notin X_{visit}$ **do**
    $X_{visit} \leftarrow X_{visit} \cup \{x'\}$
    $\tau \leftarrow$ the $\epsilon \cdot k$-th distance in C
    if $dist(x', q) < \tau$ **then**
      Add $\langle x', dist(x', q)\rangle$ into C and remove the last one from C
      $Q \leftarrow Q \cup \langle x', dist(x', q)\rangle$
  Sort Q by $dist(\cdot, q)$
$S' \leftarrow$ the first k points in C
return S'

4 提案手法

提案手法のアイデアを紹介する前に，4.1 節で任意のグラフを用いて与えられたクエリの kNN を検索するアルゴリズムについて説明する．その後，4.2 節で提案手法の基礎となる新たな定理を紹介する．4.3 節では，4.2 節で述べた理論をもとに，より実用的なアルゴリズムである LGTM を提案する．

4.1 予備知識

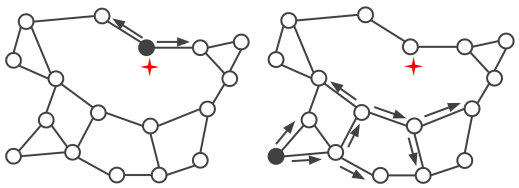
[15], [29] などで一般的に使われている貪欲な AkNN 検索アルゴリズムを説明する．$E(x)$ を点（ノード）$x \in X$ が持つエッジ集合とする．このアルゴリズムは，kNN の候補集合である C をメンテナンスする．C の要素数は最大 ϵk であり，$\epsilon \geqq 1$ はユーザが検索開始時に決めるハイパーパラメータである．一般的に ϵ が大きいほど Recall は向上するが検索速度は低下する．

はじめに，始点 s が与えられると，$\langle s, dist(s,q)\rangle$ をキュー Q に追加する．次に，Q の先頭の要素 $\langle x, \cdot\rangle$ を取り出し，エッジ $(x, x') \in E(x)$ を取得する．ただし x' は未訪問のノードである．もし $\langle x', dist(x', q)\rangle$ が現状の候補集合 C を改善する場合，これを C と Q の両方に追加する．この操作を $Q = \varnothing$ となるまで繰り返す．すなわちこのアルゴリズムは，クエリ q に接近するノードのみを訪問して現状の候補を改善し続ける．このアルゴリズムの詳細をアルゴリズム 1 に示す．

時間計算量． N を Q に追加される点の数，deg を与えられたグラフの平均次数とする．アルゴリズム 1 の時間計算量は $O(N \cdot deg)$ である．

4.2 理論

アルゴリズム 1 を高速に動作させるためには，q に近いノードを始点に選ばなければならない．それを示す例を図 1 に示す．簡単のため，図 1 では $k = 1$ および $\epsilon = 1$ に設定している．最良の場合は，図 1a のように始点が q の厳密な NN の時であり，



(a) 始点が q の NN の場合　(b) 始点が q から遠いノードの場合

図 1: アルゴリズム 1 の動作例．白の点はグラフのノード，黒の点は始点，赤の十字はクエリを表す．また，矢印はグラフを辿る向きを表す．

その時の距離計算回数（ノードの訪問回数）は最小となる．一方，始点が q から遠いノードの時，図 1b に示すように，距離計算回数が大きくなるだけでなく，局所解に陥る可能性も高くなる．したがって，高速かつ高精度に検索を行うためには，q に近いノードを始点に選ぶ必要がある．我々は，LSH を使うことによりこの課題を解決する．ユークリッド空間における LSH と近接グラフの統合が我々の提案の核であり，これにより新たな定理を発見した．

その定理を紹介する前に，LSH の詳細を説明する．

定義 2（(r, cr, p_1, p_2)-SENSITIVE HASHING [11]）． 半径 r, 近似パラメータ $c \geqq 1$，確率 p_1, p_2，$(p_1 > p_2)$，ハッシュ族 $\mathcal{H} = \{h : \mathbb{R}^d \to U\}$ において，$\forall x, x' \in \mathbb{R}^d$ に対し以下の条件が成り立つ時，$\mathcal{H}$ は (r, cr, p_1, p_2)-sensitive であるという．

- If $dist(x, x') \leqq r$, $\Pr[h(x) = h(x')] \geqq p_1$.
- If $dist(x, x') \geqq cr$, $\Pr[h(x) = h(x')] \leqq p_2$.

ユークリッド空間においてよく知られている (r, cr, p_1, p_2)-sensitive なハッシュ関数を以下に示す．

$$h(x) = \lfloor \frac{a \cdot x + b}{w} \rfloor$$

ただし a は d 次元の点であり，各次元は独立な標準正規分布 $\mathcal{N}(0, 1)$ に従う変数である．b は区間 $[0, w)$ の一様分布に従う変数であり，w はユーザが決めるハイパーパラメータである．そして，$\theta = dist(x, x')$ とした時，以下が成り立つ．

$$\Pr[h(x) = h(x')] = \int_0^w \frac{1}{\theta} \cdot f(\frac{t}{\theta}) \cdot (1 - \frac{t}{w}) dt, \quad (1)$$

$$f(z) = \frac{2}{\sqrt{2\pi}} e^{-\frac{z^2}{2}}$$

式 1 は θ が小さくなるとハッシュ関数の衝突確率が大きくなることを示す．ここで，(r, cr, p_1, p_2)-sensitive なハッシュ族は (r, c)-ball cover query を高速に処理できることに注意する．(r, c)-ball cover query の定義は以下の通り．

定義 3（(r, c)-BALL COVER QUERY）． データ集合を X，クエリを q，距離の閾値を r，近似パラメータを c，$B(q, r)$ を q を中心とする半径 r の超球とする．(r, c)-ball cover query は以下の結果を返却する．

- もし $B(q, r)$ が X の中の一つ以上の点を含む時，X の中で $B(q, cr)$ に含まれる任意の点を一つ返却する．
- もし $B(q, r)$ が X の中の点を含まない場合，何も返却しない．

m 個の独立なハッシュ関数の結合 $G(\cdot)$ を $G(x) = (h_1(x), ..., h_m(x))$ とする．$x \in X$ は $G(x)$ をキーとするバケットで管理され，ハッシュテーブルはバケットの集合である．E2LSH [11] では L 個のハッシュテーブルを構築することにより以下の性質を得る．

定理 1 [11]． $L = 1/p_1^m$ および $m = \log_{1/p_2} n$ に設定することにより，ある一定の確率で (r, c)-ball cover query の厳密解が得られ，その時間計算量は $O(n^{\frac{\ln 1/p_1}{\ln 1/p_2}})$ である．

x^* を q の NN とする．定理 2 で述べたように，LSH を用いることにより，$dist(x^*, q) \leqq r$ である時，少なくともある一定の確率で x^* を得られる．そして，その時間計算量は n に対する劣線形時間である．しかしこの結果は，必ずしも高い Recall で AkNN を得られることを意味していない．LSH によって，$dist(s, q) \leqq cr$ である始点 s は得られるが，その始点から求める kNN へのエッジがない可能性があるためである．我々はこの制限を取り払うために，$(c+1)r$-similarity graph を導入する．

定義 4（$(c+1)r$-SIMILARITY GRAPH）． データ集合を X，半径を r，近似パラメータを $c \geqq 1$ とする．$(c+1)r$-similarity graph では，各ノード $x \in X$ と $dist(x, x') \leqq (c+1)r$ である $x' \in X$ との間にのみエッジが存在する．
我々が発見した定理を次に示す．

定理 2．LSH および $(c+1)r$-similarity graph を用いることにより，厳密な NN x^*（ただし $dist(x^*, q) \leqq r$）を少なくとも一定の確率で得られ，その時間計算量は $O(\max\{n^{\frac{\ln 1/p_1}{\ln 1/p_2}}, deg\})$ である．

証明. 仮定より，$dist(x^*, q) \leqq r$．LSH を用いて，$dist(x, x^*) \leqq cr$ である点 x を少なくとも一定の確率で得る．三角不等式より，$dist(x, x^*) \leqq r + cr = (1+c)r$．定義 5 より，$x$ と x^* の間にエッジがあることが保証される．したがって，アルゴリズム 1 を用いて 1 ホップで x から x^* にアクセスできる．また，その時間計算量は定理 2 およびアルゴリズム 1 の時間計算量より，$O(\max\{n^{\frac{\ln 1/p_1}{\ln 1/p_2}}, deg\})$ である． □

さらに，q の k 番目の NN $x^{*,k}$ に対し，以下が成り立つ．

系 1． $dist(x^*, q) \leqq r$ かつ $dist(x^*, x^{*,k}) \leqq (1+c)r$ の時，LSH および $(c+1)r$-similarity graph を用いると，厳密な kNN を少なくとも一定の確率で得られ，その時間計算量は $O(\max\{n^{\frac{\ln 1/p_1}{\ln 1/p_2}}, deg\})$ である．

証明. 仮定 $dist(x^*, q) \leqq r$ および定理 3 より，少なくともある一定の確率で厳密な NN x^* を得られる．また，仮定より $dist(x^*, x^{*,k}) \leqq (1+c)r$ であるから，x^* と $x^{*,k}$ の間にはエッジが存在する．したがって，少なくともある一定の確率で，$O(1)$ のホップ数で LSH で得た始点 x から $x^{*,k}$ に到達できる． □

4.3 理論から実践

4.2 節では，$dist(x^*, q) \leqq r$ かつ $dist(x^*, x^{*,k}) \leqq (1+c)r$ の時，LSH と $(c+1)r$-similarity graph を用いると，厳密な kNN を，少なくともある一定の確率で，$O(\max\{n^{\frac{\ln 1/p_1}{\ln 1/p_2}}, deg\})$ の時間で得られることを述べた．また，一般的に $deg \ll n$ であるため，実践的には $O(\max\{n^{\frac{\ln 1/p_1}{\ln 1/p_2}}, deg^{O(1)}\}) = O(n^{\frac{\ln 1/p_1}{\ln 1/p_2}})$ であることに注意すると，(r, c)-ball cover query の処理時間と同じ時間で厳密な kNN を求められる．しかし，実際は $(c+1)r$-similarity graph には二つの制約がある．一つ目は，その構築に $O(n^2)$ の時間がかかることである．一度 $(c+1)r$-similarity graph を構築すれば任意のクエリに対して利用できるものの，大きな n に対して $O(n^2)$ の時間は実用上許容できない．二つ目は，空間計算量が最悪の場合 $O(n^2)$ となることである．したがって，これらの二つの制約を取り払いつつ，定理 3 と似た性質を持った，より実用的な AkNN 検索アルゴリズムが必要である．ここで重要となるのは，定理 2 で述べた LSH の性質より，ハッシュテーブルの数を増やせば AkNN の精度が向上することである．

そこで我々は LGTM（Local Graph Traversal Method）を提案する．LGTM は 4.2 節で述べたアイデアを採用しつつ，より実用的なアルゴリズムとなっている．

4.3.1 前 処 理

まず，LGTM の前処理について説明する．この前処理は一度だけ行う．詳細はアルゴリズム 2 に示す．

LSH とサンプリング． LGTM のアイデアの一つは，サンプリングにより，検索時の $O(n^{\frac{\ln 1/p_1}{\ln 1/p_2}})$ の時間を減らすことである．LGTM の前処理では，L 個のハッシュテーブルを作る．E2LSH とは異なり，各バケットは定数個の点を格納する [1]．これにより，クエリが発行された時，始点を決定するのにかかる時間は $O(1)$ となる．また，$L = O(1)$ と設定することにより，ハッシュテーブルの空間計算量と構築にかかる時間計算量は共に $O(n)$ となる．

無向 AKNNG の利用． 二つ目のアイデアは，近似 KNN グラフ（AKNNG）を利用することである（KNN グラフは構築に $O(n^2)$ の時間がかかるため，近似して使う）．ここで，K はグラフの次数であり，検索結果の数 k とは無関係であることに注意する．$K = deg = O(1)$ である．したがって，LGTM のインデックス（ハッシュテーブルおよび AKNNG）の空間計算量は $O(n)$ である．

AKNNG は，各ノード $x \in X$ とその AKNN である $x' \in X$ の間にのみエッジを作る．[13] の AKNNG 構築アルゴリズムを利用することにより，$O(n)$ に近い時間で AKNNG を得られる．その後，各エッジを無向にすることで，無向 AKNNG を得る．

4.3.2 オンライン（クエリ）処理

簡単のため，はじめにハッシュテーブルが一つ（$L = 1$）の場合について説明した後，L が一般（$L \geqq 1$）の場合について説明する．

一つのハッシュテーブルを用いる場合． クエリ q が与えられると，まず $G(q)$ を計算し，$G(q)$ のバケット内における q の NN s を計算する．ハッシュテーブルに $G(q)$ をキーとするバケットが無い場合は，s をランダムに決定する．次に，s を始点とし，

1：我々の実装では，各バケットが格納する点の数を 50 個に設定した．

Algorithm 2: PRE-PROCESSING OF LGTM

Input: X, L, and m

/* Hash table construction */
for *each* $x \in X$ **do**
  for *each* $i \in [1, L]$ **do**
    Maintain x in the bucket with $G_i(x) = (h_{i,1}(x), ..., h_{i,m}(x))$
for *each* $i \in [1, L]$ **do**
  for *each bucket in the* i*-th hash table* **do**
    Sample $O(1)$ points in this bucket
    Remove not sampled points from this bucket
/* Undirected AkNNG construction */
Run NNDESCENT [13] on X
Convert each directed edge into undirected one

Algorithm 3: QUERY-PROCESSING OF LGTM (a single hash table case)

Input: X, q, k, ϵ, a hash table, and an AkNNG

$s \leftarrow$ the nearest neighbor of q in the bucket with $G(q)$
Run Algorithm 1

AkNNG 上でアルゴリズム 1 を実行する．

議論． 系 2 と異なり，LGTM はアルゴリズム 2 でサンプリングを行い，AkNNG を用いているため，s から q の kNN に到達できる保証はない．したがって，より近いノードを始点として設定できる確率を高める必要がある．ここで，ハッシュ関数の衝突確率を $col_i(x, x') = \Pr[h_i(x) = h_i(x')]$ とすると，一つのバケットが x と x' の両方を格納している確率は以下のように表せる．

$$\Pr[G(x) = G(x')] = \prod_{i=1}^{m} col_i(x, x').$$

それぞれのハッシュテーブルは独立に構築されるため，上式より，任意のハッシュテーブルのバケットが x と x' の両方を格納している確率は以下のように表せる．

$$\Pr[\exists j \in [1, L], G_j(x) = G_j(x')] = L \prod_{i=1}^{m} col_i(x, x'), \quad (2)$$

G_i は i 番目のハッシュ関数の結合である．ここで，L 個のハッシュテーブルを使えば，L 個の始点が得られることに留意すると，式 2 は，L に比例して q に近い始点を得られる確率が向上することを示している．すなわち，LGTM はその後 AkNNG を使ってさらに q に近い点に到達できる確率が向上する．したがって，L が大きくなると Recall が向上する確率が上がる．これが LGTM において複数のハッシュテーブルを用いるモチベーションとなる．従来の state-of-the-art のアルゴリズム [15], [29] は始点が固定されているため，この恩恵を受けられない．

これを LGTM で実現するため，アルゴリズム 3 を L 回実行するというアプローチが考えられる．しかし，単純に L 回実行すると速度が低下する．そこで，我々はマルチスレッディングによりこの課題を解決する．クエリ q が発行されると，クエリ

Algorithm 4: QUERY-PROCESSING OF LGTM

Input: X, q, k, ϵ, t hash tables (t is the number of available threads), and an AkNNG

Prepare t copies of q, q_1, ..., q_t
Run Algorithm 3 for q_1, ..., q_t in parallel
Merge the AkNN results of q_1, ..., q_t

が L 個コピーされ，それぞれ異なる始点からクエリに接近すると考える．そして，各クエリはマルチスレッディングにより並列に処理され，最後に各スレッドで計算した AkNN をマージする．この方法により，より多くのスレッドがあれば，速度を低下させることなく Recall を向上できる．

複数のハッシュテーブルを用いる場合． t を利用可能なスレッド数とする（前処理における L を $L \geqq t$ とする）．はじめにクエリ q が与えられると，t 個の q のコピー $q_1, ..., q_t$ を作る．次に，クエリのコピーごとにアルゴリズム 3 を並列に実行する．ここで，それぞれのコピー q_i に対して $G_i(\cdot)$ を使うことに注意する．これにより，異なるハッシュテーブルから始点が得られる．最後に，各コピーの AkNN をマージして，AkNN を得る．

時間計算量． ハッシュテーブルの各バケットは定数個のノードを格納しているため，始点は $O(1)$ で得られる．N_i をクエリのコピー q_i に対して実行したアルゴリズム 1 の中で Q に追加したノード数とする．クエリのコピーは並列処理されるため，アルゴリズム 4 の時間計算量は $O(\max_{i \in [1,t]} N_i \cdot deg) = O(\max_{i \in [1,t]} N_i)$ である．

5 実　　験

本章では，提案手法の実験結果を示す．全ての実験は，Ubuntu 16.04 LTS OS，18 core 3.0GHz Core i9-9980XE，128GB RAM を搭載した計算機上で行った．

5.1 設　　定

データセット． 次の三つのデータセットを実験に用いた．

- SIFT[2]：128 次元，1,000,000 個の SIFT 特徴量記述子．
- GIST：960 次元，1,000,000 個の GIST 特徴量記述子．
- Deep [7]：96 次元，10,000,000 個の Deep 特徴量．

クエリには各データセットに用意されているものを用いた（SIFT および GIST は 10,000 個，Deep は 1,000 個）．本章で述べる実験結果は，全てのクエリを実行した時の平均値である．

評価したアルゴリズム． 次のアルゴリズムを評価した．

- LGTM：本稿で提案するアルゴリズム．LGTM の LSH は，SIFT，GIST，Deep を用いた実験に対し，m（w）をそれぞれ 4（200），5（1），10（1）に設定した．また，ハッシュテーブルは 18 個構築した．
- HNSW [29]：small world network モデルのグラフインデックスを用いた state-of-the-art の AkNN 検索アルゴリズム．
- NSG [15]：単調経路を AkNNG に追加したグラフインデックスを用いた state-of-the-art の AkNN 検索アルゴリズム．

2：http://corpus-texmex.irisa.fr/

- AKNNG：常に始点をランダムに決定し，無向 AkNNG 上で実行したアルゴリズム 1．このアルゴリズムは，LGTM の始点の決定方法の有効性を示すために評価する．

[15] および [29] において，HNSW および NSG が最高性能と示されており，[26] において，グラフインデックスを用いたアルゴリズムが他のアプローチと比べて遥かに高速かつ高精度であることが示されているため，上で述べたアルゴリズム以外のアルゴリズムは評価しない．上で述べたアルゴリズムは C++で実装し，g++ 5.5 を用いて，最適化オプション-O3 を付加してコンパイルした．また，マルチスレッディングには OpenMP を用いた．

5.2 t の影響

はじめに，使用するハッシュテーブル（スレッド）の数が変化した時の Recall の変化を確認する．この実験では，$k = 10, \epsilon = 1$ とした．また，SIFT および Deep では $deg = 15$，GIST では $deg = 30$ とした．

t を増加させた時の平均距離計算回数[3]と Recall の変化を図 2 に示す．図 2 より，全てのデータセットにおいて，t が大きくなるほど Recall が大きくなることがわかる．この結果は，4.3.2 項における議論と一致する．一方，t が大きくなるほど距離計算回数も大きくなることがわかる．これは，t が大きくなるほど $\max_{i \in [1,t]} N_i$ も大きくなるためである．これらの結果は，t により速度と精度を調整できることを意味している．

5.3 AKNNG との比較

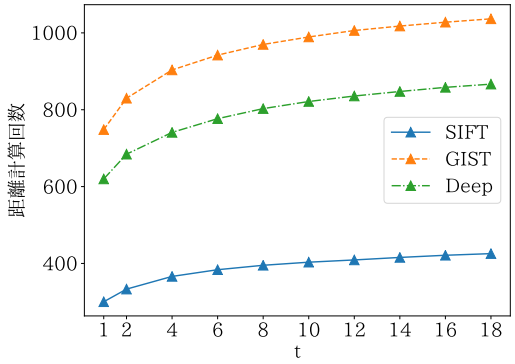
ハッシュの効果を検証するため，LGTM と AKNNG の比較を行った．AKNNG は LGTM と同じ並列処理を行っていることに注意する．$k = 10, t = 8$ に設定した時の両者の距離計算回数と Recall を表 1 に示す．表 1 より，LGTM は AKNNG よりも距離計算回数が少ないにもかかわらず，Recall が高いことがわかる．これは，我々が提案した始点の決め方により，ランダムよりも高速かつ高精度な結果を得られることを表している．

5.4 State-of-the-art のアルゴリズムとの比較

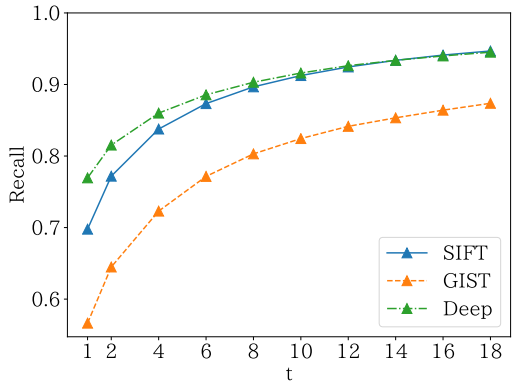
次に，LGTM と HNSW および NSG との比較を行う．LGTM ($t = 1, 8, 18$)，HNSW，および NSG における速度と精度のトレードオフを図 3 に示す．図 3a から 3c は実行時間と Recall のトレードオフを，図 3d から 3f は距離計算回数と Recall のトレードオフを表す曲線である．実行時間と Recall および距離計算回数と Recall は同じ傾向を示している．

概観. 図 3 より，LGTM は同じ距離計算回数でも HNSW および NSG よりも精度の高い kNN を計算していることがわかる．この結果は，LSH と AkNNG を統合したインデックスの方が，他の複雑なグラフ構造より高性能であることを示している．

図 3e および図 2 より，GIST データセットでは，他のデータセットよりも同じ距離計算回数でも Recall が低い．これは，



(a) 距離計算回数



(b) Recall

図 2: t を変えた時の距離計算回数と Recall の変化

表 1: AKNNG と LGTM の比較

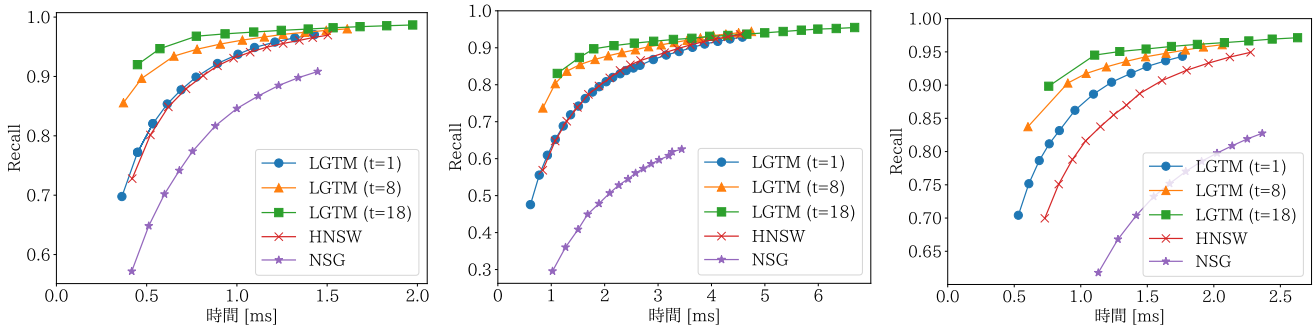
Data	アルゴリズム	距離計算回数	Recall
SIFT	AKNNG	397	0.818
	LGTM	**391**	**0.889**
GIST	AKNNG	974	0.722
	LGTM	**970**	**0.803**
Deep	AKNNG	846	0.848
	LGTM	**802**	**0.903**

データの分布の歪みが大きく，頻繁に kNN の候補集合が更新されるためと考えられる．それにも関わらず，LGTM ($t = 18$) は他のアルゴリズムよりも少ない距離計算回数で正確な kNN を計算できていることがわかる．

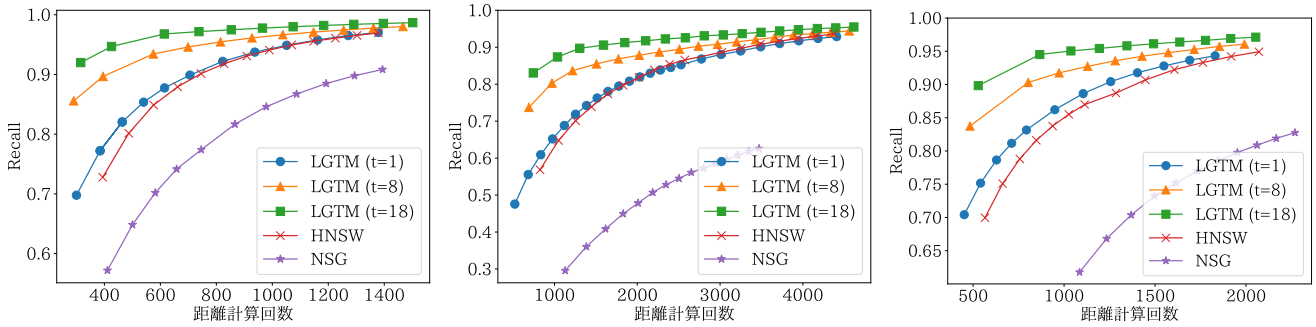
NSG との比較. NSG と他のアルゴリズムを比較する．図 3 は，NSG の性能が他のアルゴリズムの性能よりも低いことを表している．NSG は，始点を常にデータ空間の重心の最近傍に設定しているため，始点とクエリが遠いことが多く，多くの距離計算を必要とする．さらに，始点とクエリが遠いと，アルゴリズム 1 により局所解に陥り，Recall が低下しやすい．これが NSG の性能が他の手法に比べて低い原因だと考えられる．同時に，この結果は，高速に高精度な AkNN を検索するためには，q の近くの始点を選ぶ必要があることを意味している．

HNSW との比較. LGTM と HNSW に焦点を当てる．LGTM は，ハッシュテーブルの数が一つであっても，HNSW と同等かそれ以上の性能を達成している．これは，HNSW よりも

3：LGTM は，1 個のクエリに対し，クエリのコピーを t 個作り，それらを並列に処理する．そのため，ここでの平均距離計算回数は，各スレッドで発生した距離計算回数の最大値を全クエリにわたって平均したものを指す．



(a) SIFT（実行時間 vs. Recall）　(b) GIST（実行時間 vs. Recall）　(c) Deep（実行時間 vs. Recall）



(d) SIFT（距離計算回数 vs. Recall）　(e) GIST（距離計算回数 vs. Recall）　(f) Deep（距離計算回数 vs. Recall）

図 3: t を変化させた LGTM と HNSW および NSG との比較．上図は実行時間と Recall のトレードオフを，下図は距離計算回数と Recall のトレードオフを表す曲線である．左上に位置するものが高速かつ高精度であることを意味する．

表 2: LGTM，HNSW，および NSG における k の影響

		$k=10$		$k=20$		$k=30$		$k=40$	
データ	アルゴリズム	距離計算回数	Recall	距離計算回数	Recall	距離計算回数	Recall	距離計算回数	Recall
SIFT	NSG	411	0.572	582	0.681	730	0.735	868	0.768
	HNSW	394	0.727	576	0.807	745	0.846	909	0.870
	LGTM ($t=8$)	**391**	**0.889**	**551**	**0.889**	**693**	**0.891**	**858**	**0.896**
GIST	NSG	1129	0.296	1617	0.394	2009	0.452	2347	0.490
	HNSW	1003	0.632	1446	0.706	1830	0.747	2207	0.775
	LGTM ($t=8$)	**970**	**0.803**	**1408**	**0.798**	**1795**	**0.802**	**2156**	**0.809**
Deep	NSG	910	0.538	1234	0.650	1498	0.704	1801	0.744
	HNSW	805	0.784	1158	0.841	1434	0.864	1731	0.882
	LGTM ($t=8$)	**803**	**0.903**	**1129**	**0.906**	**1429**	**0.912**	**1715**	**0.917**

LGTM の始点の決め方の方がより効果的であることを意味している．また，18 スレッドを使ったときは，明らかに LGTM の方が優れている．例えば，LGTM は HNSW よりも速く 90% の Recall に到達している．このマルチスレッディングのアプローチは，LGTM の利点の一つである．NSG や HNSW のような複雑なグラフ構造は，始点を固定して構築する必要がある．このようなグラフでは，マルチスレッディングによる恩恵を受けられない．したがって，我々のシンプルかつ理論的に設計されたアプローチが他のアルゴリズムよりも優れている．

5.5 k の影響

LGTM，HNSW，および NSG における k を変化させた時の影響を調べた．その結果を表 2 に示す．LGTM は，k が変化しても，他のアルゴリズムよりも少ない距離計算回数で高い Recall を実現している．また，LGTM は k の変化に対し頑健であり，k が大きくなっても高い性能を保っていることがわかる．そして，図 2 に示すように，スレッド数（ハッシュテーブルの数）を大きくすることで容易に Recall を向上することができる．

6 結　　論

本論文では，高次元ユークリッド空間における AkNN 検索アルゴリズムを提案した．我々は，LSH と近接グラフを組み合わせることにより，ある一定の確率で，n に対する劣線形時間で正確な kNN を得られることを保証する定理を発見した．しかし，この方法は，理論的には有効であるものの，実用上許容

できない制約を抱えている．そこで，より実用的なアルゴリズムである LGTM を提案した．LGTM は，従来の手法とは異なり，速度と精度のトレードオフを調節するパラメータを持っており，その特性は理論的に保証されている．さらに，実世界のデータセットである SIFT，GIST，および Deep を用いた実験により，LGTM の性能は既存の state-of-the-art よりも優れていることを示した．

謝辞．本研究の一部は，文部科学省科学研究費補助金・基盤研究 (A)(18H04095) および JST さきがけ（JPMJPR1931）の支援を受けたものである．

文　献

[1] Andoni, A., Indyk, P.: Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In: FOCS. pp. 459–468 (2006)
[2] Andoni, A., Razenshteyn, I.: Optimal data-dependent hashing for approximate near neighbors. In: STOC. pp. 793–801 (2015)
[3] Arya, S., Mount, D.M.: Approximate nearest neighbor queries in fixed dimensions. In: SODA. vol. 93, pp. 271–280 (1993)
[4] Babenko, A., Lempitsky, V.: The inverted multi-index. IEEE transactions on pattern analysis and machine intelligence 37(6), 1247–1260 (2014)
[5] Babenko, A., Lempitsky, V.: Aggregating local deep features for image retrieval. In: ICCV. pp. 1269–1277 (2015)
[6] Babenko, A., Lempitsky, V.: Tree quantization for large-scale similarity search and classification. In: CVPR. pp. 4240–4248 (2015)
[7] Babenko, A., Lempitsky, V.: Efficient indexing of billion-scale datasets of deep descriptors. In: CVPR. pp. 2055–2063 (2016)
[8] Beckmann, N., Kriegel, H.P., Schneider, R., Seeger, B.: The r*-tree: an efficient and robust access method for points and rectangles. In: SIGMOD. pp. 322–331 (1990)
[9] Bentley, J.L.: Multidimensional binary search trees used for associative searching. Communications of the ACM 18(9), 509–517 (1975)
[10] Cheng, J., Leng, C., Wu, J., Cui, H., Lu, H.: Fast and accurate image matching with cascade hashing for 3d reconstruction. In: CVPR. pp. 1–8 (2014)
[11] Datar, M., Immorlica, N., Indyk, P., Mirrokni, V.S.: Locality-sensitive hashing scheme based on p-stable distributions. In: SoCG. pp. 253–262 (2004)
[12] Dearholt, D., Gonzales, N., Kurup, G.: Monotonic search networks for computer vision databases. In: Twenty-Second Asilomar Conference on Signals, Systems and Computers. vol. 2, pp. 548–553 (1988)
[13] Dong, W., Moses, C., Li, K.: Efficient k-nearest neighbor graph construction for generic similarity measures. In: WWW. pp. 577–586 (2011)
[14] Fu, C., Cai, D.: Efanna: An extremely fast approximate nearest neighbor search algorithm based on knn graph. arXiv preprint arXiv:1609.07228 (2016)
[15] Fu, C., Xiang, C., Wang, C., Cai, D.: Fast approximate nearest neighbor search with the navigating spreading-out graph. PVLDB 12(5), 461–474 (2019)
[16] Ge, T., He, K., Ke, Q., Sun, J.: Optimized product quantization. IEEE transactions on pattern analysis and machine intelligence 36(4), 744–755 (2013)
[17] Ge, T., He, K., Ke, Q., Sun, J.: Optimized product quantization for approximate nearest neighbor search. In: CVPR. pp. 2946–2953 (2013)
[18] Harwood, B., Drummond, T.: Fanng: Fast approximate nearest neighbour graphs. In: CVPR. pp. 5713–5722 (2016)
[19] He, X., Wang, P., Cheng, J.: K-nearest neighbors hashing. In: CVPR. pp. 2839–2848 (2019)
[20] Heo, J.P., Lee, Y., He, J., Chang, S.F., Yoon, S.E.: Spherical hashing. In: CVPR. pp. 2957–2964 (2012)
[21] Heo, J.P., Lin, Z., Shen, X., Brandt, J., Yoon, S.E.: Shortlist selection with residual-aware distance estimator for k-nearest neighbor search. In: CVPR. pp. 2009–2017 (2016)
[22] Jegou, H., Douze, M., Schmid, C.: Product quantization for nearest neighbor search. IEEE transactions on pattern analysis and machine intelligence 33(1), 117–128 (2010)
[23] Jin, Z., Zhang, D., Hu, Y., Lin, S., Cai, D., He, X.: Fast and accurate hashing via iterative nearest neighbors expansion. IEEE transactions on cybernetics 44(11), 2167–2177 (2014)
[24] Kulis, B., Grauman, K.: Kernelized locality-sensitive hashing for scalable image search. In: ICCV. pp. 2130–2137 (2009)
[25] Lempitsky, V.: The inverted multi-index. In: CVPR. pp. 3069–3076 (2012)
[26] Li, W., Zhang, Y., Sun, Y., Wang, W., Li, M., Zhang, W., Lin, X.: Approximate nearest neighbor search on high dimensional data-experiments, analyses, and improvement. IEEE Transactions on Knowledge and Data Engineering 32(8), 1475–1488 (2020)
[27] Liu, W., Wang, J., Kumar, S., Chang, S.F.: Hashing with graphs. In: ICML. pp. 1–8 (2011)
[28] Lowe, D.G.: Distinctive image features from scale-invariant keypoints. International journal of computer vision 60(2), 91–110 (2004)
[29] Malkov, Y.A., Yashunin, D.: Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. IEEE Transactions on Pattern Analysis and Machine Intelligence 42(4), 824–836 (2020)
[30] Malkov, Y., Ponomarenko, A., Logvinov, A., Krylov, V.: Approximate nearest neighbor algorithm based on navigable small world graphs. Information Systems 45, 61–68 (2014)
[31] Norouzi, M., Fleet, D.J.: Cartesian k-means. In: CVPR. pp. 3017–3024 (2013)
[32] Sun, Y., Wang, W., Qin, J., Zhang, Y., Lin, X.: Srs: Solving c-approximate nearest neighbor queries in high dimensional euclidean space with a tiny index. PVLDB 8(1) (2014)
[33] Wang, J., Wang, J., Yu, N., Li, S.: Order preserving hashing for approximate nearest neighbor search. In: MM. pp. 133–142 (2013)
[34] Weber, R., Schek, H.J., Blott, S.: A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In: VLDB. pp. 194–205 (1998)
[35] Xia, Y., He, K., Wen, F., Sun, J.: Joint inverted indexing. In: ICCV. pp. 3416–3423 (2013)