情報処理学会論文誌 Vol.62 No.4 1086–1098 (Apr. 2021)

高次元データに対するグラフインデックスを用いた近似範囲検索アルゴリズム

新井 悠介[1,a)] 天方 大地[1,2,b)] 原 隆浩[1,c)] 藤田 澄男[3,d)]

受付日 2020年6月24日, 採録日 2021年1月12日

概要：範囲検索は，データベース，情報検索，情報推薦，およびデータマイニングなど，幅広い分野で応用されており，その高速化は重要な課題である．また，近年では画像および音声をはじめとする高次元データを扱うアプリケーションが増えている．しかし高次元データの範囲検索は，次元の呪いにより，木構造を用いた従来手法では高速化が困難である．また，高次元データを低次元空間に写像する近似手法では，利用する距離関数によっては写像の設計が困難である．近年，次元削減を行わないデータ構造として，グラフを用いたインデックスが注目されている．本論文では，ユークリッド空間におけるグラフを用いたインデックスをメトリック空間に拡張し，任意の距離関数の利用を可能にする．さらに，グラフを用いた効率的な近似範囲検索アルゴリズムを提案する．実データを用いた実験により，提案アルゴリズムの有効性を示す．

キーワード：グラフインデックス，範囲検索

An approximate Range Search Algorithm with a Graph-based Index on High-dimensional data

Yusuke Arai[1,a)] Daichi Amagata[1,2,b)] Takahiro Hara[1,c)] Sumio Fujita[3,d)]

Received: June 24, 2020, Accepted: January 12, 2021

Abstract: Range search is a primitive operator in databases, information retrieval, information recommendation, and data mining. Recently, many applications have been generating high dimensional data. Although they require fast similarity search in any metric space, exact and efficient range search on high-dimensional data is difficult and existing tree-based indices suffer from the curse of dimensionality. Dimensionality reduction based algorithms have been proposed, but they are not available on metric space. Proximity graph-based indices have recently shown superior performances on high-dimensional data w.r.t both efficiency and accuracy. In this paper, we extend a state-of-the-art graph-based index, which can be built only on the Euclidean space, so that we can utilize it on any metric space. Furthermore, we propose a novel approximate range search algorithm with a graph-based index. Our experiments on real datasets confirm that our algorithm provides a significantly better efficiency compared with state-of-the-art while keeping almost perfect results.

Keywords: graph-based index, range search

[1] 大阪大学大学院情報科学研究科
Graduate School of Information Science and Technology, Osaka University, Suita, Osaka 565–0871, Japan
[2] JST さきがけ
JST PRESTO, Chiyoda, Tokyo 102–0076, Japan
[3] ヤフー株式会社
Yahoo Japan Corporation, Chiyoda, Tokyo 102–8282, Japan
[a)] arai.yusuke@ist.osaka-u.ac.jp
[b)] amagata.daichi@ist.osaka-u.ac.jp
[c)] hara@ist.osaka-u.ac.jp
[d)] sufujita@yahoo-corp.jp

1. はじめに

範囲検索は，ある距離関数が定義されている空間において，クエリ q と検索範囲 r が与えられたとき，与えられたデータ集合の中から，クエリとの距離が r 未満であるデータを検索する操作である．範囲検索は，データベース [6]，情報検索 [29]，情報推薦 [21], [30]，データマイニング [9], [26] など，多くの分野で応用されている．範囲検

© 2021 Information Processing Society of Japan

索を実現する最も簡単な方法は，与えられたデータ集合の各要素とクエリとの距離をすべて計算するという方法である．しかしこの方法では，大量のデータに対して検索を行う際の実行時間が非常に大きくなるという問題点がある．近年，画像，映像，および音声をはじめとした高次元データを扱うアプリケーションが増えている．このようなアプリケーションで生成されるデータの類似度を測定するための関数は，ユークリッド距離に限らず l_p 距離や角距離などが存在し，アプリケーションが求める類似度関数を入力できるアルゴリズムが求められる．本論文ではメトリック空間に焦点を当てる．木構造を用いた正確な範囲検索アルゴリズム [4], [32] では，次元の呪いと呼ばれる現象により高速化ができない．この問題を解決するため，ハッシュ [7], [13], [17], [22], [23], [27], [33] や量子化 [12], [18], [20] による近似手法が提案されている．しかし，これらの手法は利用できる距離関数が限定的であり，メトリック空間における範囲検索を高速化するハッシュ関数や量子化器を設計するのは困難である．また近年，高次元空間における近似 k 最近傍検索の研究において，近接グラフを用いたアプローチ [10], [11], [15], [16], [24], [25] が，ハッシュや量子化による方法よりも優れた性能を示している．

本論文では，この観測に基づいて，グラフインデックスを用いた新しい近似範囲検索アルゴリズムを提案する．提案アルゴリズムでは，各データをノード，データどうしの関係をエッジとして表現し，エッジをたどる操作を繰り返すことにより範囲検索を行う．これにより，任意の距離関数を利用可能にし，高次元データに対して高速な近似範囲検索を実現する．

1.1 研究動機

メトリック空間における高次元データの高速な近似範囲検索が要求されるアプリケーションは数多く存在する．

例 1. あるECサイトにおいて，ユーザの嗜好を考慮して商品の推薦を行う状況を考える．このときサーバはユーザが過去に閲覧した商品のデータをクエリ q に，距離 r を検索範囲として，扱っている商品のデータ全体に対して範囲検索を行うことで，類似した商品をユーザに推薦する．

例 2. リアルタイム翻訳システムにおいて，音声を認識し，話した単語を推測する機能を考える．このシステムでは，入力された音声の特徴量をクエリに，距離 r を検索範囲として，あらかじめ登録している単語の発音の特徴量に対して範囲検索を行う．検索結果として返却された単語のうち，文脈に最も合う単語を出力する．

多くのシステムにおいて，商品データや音声の特徴量は高次元データとして表現される．ECサイトや翻訳システムでは低遅延なレスポンスが求められるため，高次元データに対する高速なアルゴリズムが必要となる．さらに，アプリケーションによっては，一般的に用いられる距離関数であるユークリッド距離ではなく，角距離などの距離関数を用いることによりユーザから見て高品質な結果が得られる可能性がある．実際に，文献 [3], [14], [28] では，データ間の角度を考慮した推薦アルゴリズムが提案されている．したがって，先述したように，任意の距離関数を利用可能なインデックスが求められる．

例 3. 文献 [9], [26] では，密度準拠クラスタリングという手法を提案している．密度準拠クラスタリングは非線形なクラスタ境界を表現できるため，注目を集めている．文献 [9], [26] で提案している密度準拠クラスタリングには，与えられたデータ集合に対して範囲検索を行い，各データの周辺にあるデータを数え上げる操作が含まれる．

データ集合の要素数が多いときは，範囲検索の実行時間が長くなるため，データ分析の作業を妨げる可能性がある．したがって，高速なクラスタリングを実現するには，範囲検索の高速化が必要である．

1.2 提案アルゴリズムの概要

本論文では，NSG [11] と呼ばれるユークリッド空間における近似 k 最近傍検索のためのグラフインデックスをメトリック空間に拡張し，任意の距離関数を利用可能にする．さらに，グラフを用いた範囲検索アルゴリズムを提案する．提案アルゴリズムは，クエリとの距離が大きいノードに対する無駄なアクセスを減らしつつ検索範囲内のデータを1つ発見する Global Range Search と，検索範囲内のデータを貪欲に検索する Local Range Search で構成される．Global Range Search では，遠回りの起こらないパスをたどってクエリに接近し，Local Range Search では検索範囲内のパスを貪欲にたどることにより効率的な範囲検索を実現する．

1.3 貢献

以下に本論文の貢献を示す．

(1) NSG をメトリック空間に拡張する．
(2) グラフを用いた効率的な近似範囲検索アルゴリズムを提案する．
(3) 実データを用いた実験により，提案アルゴリズムの有効性を確認する．

1.4 本論文の構成

以下では，2章で本論文で用いる用語を定義し，3章で関連研究を紹介する．4章で本論文の提案アルゴリズムについて説明し，5章で性能評価実験の結果を示す．

2. 予備知識

本論文で用いる用語を定義する．

定義 1（メトリック空間）. 集合 X 上で定義された2変数

© 2021 Information Processing Society of Japan

の実数値関数 $distance : X \times X \to \mathbb{R}$ が，x, y, z を X の任意の元として，

(1) $distance(x, y) \geq 0$（非負性）
(2) $x = y \Leftrightarrow distance(x, y) = 0$（非退化性）
(3) $distance(x, y) = distance(y, x)$（対称性）
(4) $distance(x, y) + distance(y, z) \geq distance(x, z)$（三角不等式）

を満たすとき，$distance$ は距離関数あるいは単に距離およびメトリックであるといい，対 $(X, distance)$ を距離空間あるいはメトリック空間という．

定義 2（範囲検索）．メトリック空間 $\mathcal{M} = (X, distance)$ において，クエリ $q \in X$，検索範囲 $r \in \mathbb{R}$，およびデータ集合 $P = \{p_1, p_2, \ldots, p_N\} \subset X$ が与えられたとする．範囲検索は，$\{p \in P \mid distance(p, q) < r\}$ を解として返却する操作である．

高次元空間における正確な範囲検索を高速に行うことは困難であるため [31]，本論文では近似範囲検索アルゴリズムを設計する．

3. 関連研究

これまでに多くの範囲検索アルゴリズムが提案されている．本章では，木構造とハッシュを用いた範囲検索を紹介する．その後，グラフを用いた高速な近似 k 最近傍検索アルゴリズムについて述べる．

3.1 木構造を用いた範囲検索

文献 [4] では，2 分探索木を k 次元ユークリッド空間に拡張した k-d 木と呼ばれるデータ構造を提案している．k-d 木は，データ空間を，1 つの軸に直交する超平面で 2 つに分割する．この操作を再帰的に行うことでインデックスを構築する．検索時は，検索範囲と重なる分割領域に含まれるデータのみを調べることにより枝刈りができる．

文献 [32] では，VP 木と呼ばれるデータ構造を提案している．VP 木は，Vantage Point $p \in P$ と閾値 t を選び，データを p との距離が t 未満であるものと t 以上であるものに分割する．この操作を再帰的に行うことでインデックスを構築する．検索時は，三角不等式を用いて距離を計算すべきデータを絞り込むことができるため，効率的な範囲検索を実現する．また VP 木のデータ分割方法はユークリッド空間に依存しないため，メトリック空間において利用できる．文献 [5] において，VP 木はメトリック空間における木構造で最も性能が良いことが示されている．

しかし高次元空間では枝刈り効率が下がるため，木構造によるインデックスを用いて高速に範囲検索を実行することはできない．

3.2 LSH：Locality Sensitive Hashing

文献 [13] では，LSH というハッシュ関数およびそれを用いた近似範囲検索アルゴリズムが提案されている．LSH は近いデータどうしが高確率で衝突するハッシュ関数である．データ集合のすべての要素のハッシュ値を計算し，ハッシュテーブルに格納することでインデックスを構築する．検索時は，クエリに対してハッシュ値を計算し，同じハッシュ値を持つデータを取得し，その中で範囲検索を行うことで，距離計算すべき点を絞り込むことができる．

しかし LSH を用いると，解となるデータが他のバケットに格納されることがある．そのため，ハッシュテーブルを複数個用意して精度を向上するという方法が用いられるが，これによってメモリ消費量が増大するだけでなく，距離計算の回数も増えるため，実行時間も増大する．また，ハッシュ関数は使用する距離関数ごとに設計しなければならないという問題点がある．

この問題を解決するために，文献 [23] では Multi-Probe LSH という手法を提案している．LSH を利用すると，解となるデータとクエリが異なるバケットに格納される場合であっても，隣接したバケットに入る可能性が高い．Multi-Probe LSH では，この性質を利用して隣接したバケットのうち解が含まれる可能性の高いバケットを選択する．これにより，メモリ消費量を抑えて精度を向上できる．文献 [27] では，SRS という手法を提案している．SRS は，ハッシュテーブルと木構造を組み合わせてデータを管理する．検索時は，元のデータ空間における距離と LSH によって写像された空間における距離の比がカイ 2 乗分布に従うことを利用し，木構造を用いて解が含まれる可能性の高いバケットを効率的に選択する．SRS も Multi-Probe LSH と同様に，多くのハッシュテーブルを用意することなく，精度を向上できる．

しかし，Multi-Probe LSH や SRS などの手法では，精度を高めるために多くのバケットを選択すると，距離計算の回数が増加するため実行時間が著しく大きくなる．また，LSH と同様に，検索に使用する距離関数に応じてハッシュ関数やバケットの選択方法を設計しなければならないため，メトリック空間で LSH を利用できない．文献 [1], [7], [33] では，l_p 空間で利用可能な様々なハッシュ関数を提案しているが，メトリック空間で利用可能なハッシュ関数はこれまでに提案されていない．

3.3 グラフインデックスを用いた近似 k 最近傍検索

グラフは，データ空間を分割するのではなく，隣接関係によりデータを管理するため，近年，高次元データに対する近似近傍検索のためのデータ構造として有望視されている [10], [11], [15], [16], [24], [25]．グラフインデックスを用いた近似範囲検索アルゴリズムはこれまでに研究されていないが，近似 k 最近傍検索についてはさかんに研究されている．本節では，近接グラフを用いた近似 k 最近傍検索アルゴリズムとそれを実現するグラフインデックスについて

© 2021 Information Processing Society of Japan

Algorithm 1: KNNSearch

Input: query q, number of result k, start node s and candidate set size l
Output: approximate k nearest neighbors

```
candidate set C ← {s}
while true do
    i ← the index of the first unvisited node in C
    if i ≥ l then
        break
    v ← i-th candidate
    mark v as visited
    for ∀u ∈ v.neighbors do
        C ← C ∪ {u}
    sort C in ascending order of the distance to q
    if |C| > l then
        C.resize(l)
return top k candidates in C
```

説明する．まず，k 最近傍検索の定義を以下に示す．

定義 3（k 最近傍検索）．メトリック空間 $\mathcal{M}$ において，クエリ q，検索結果の数 k，およびデータ集合 $P = \{p_1, p_2, \ldots, p_N\}$ が与えられたとする．k 最近傍検索は，P の中で q との距離が最も小さい k 個のデータを返却する．

近接グラフを用いた近似 k 最近傍検索アルゴリズムを Algorithm 1 [11] に示す．入力として，クエリ q，検索結果の数 k，および候補集合の最大要素数 l を与える．まず，始点を候補集合 C に追加する．その後，以下の操作を C が更新されなくなるまで行う．C からノード v を取り出し，v の隣接ノードを C に追加する．次に，C の要素を，q との距離でソートする．ここで，$|C| > l$ ならば $|C| = l$ となるまで C をリサイズする．C が更新されなくなれば，C の上位 k 個を結果として返却する．次に，近似 k 最近傍検索を実現する近接グラフについて説明する．

3.3.1 AkNNG：Approximate kNN Graph

代表的な近接グラフは，各ノードと k 最近傍との間にエッジを作る kNNG：kNN Graph である．3NNG の例を**図 1** に示す．kNNG の構築には $O(N^2)$ の時間計算量がかかるため，これを近似した AkNNG の高速な構築アルゴリズムが提案されている [8], [10], [19]．AkNNG をインデックスとして用いると，高次元データに対して高速に近似 k 最近傍検索を実行できる [15]．しかし，クラスタ状に分布したデータに対して AkNNG を構築すると，非連結なグラフが作られる可能性があるため，クラスタ外のクエリに対して検索を行うことが困難な場合がある．

3.3.2 NSW：Navigable Small World graphs

文献 [24] では，NSW というグラフインデックスを提案している．NSW にデータを追加するためには，現時点で構築されている NSW を利用して近似 k 最近傍検索を行い，

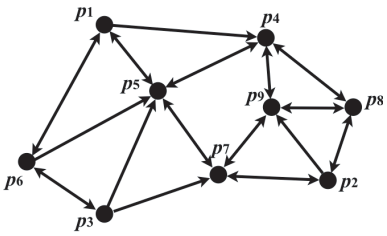


図 1 3NNG の例
Fig. 1 Example of 3NNG.

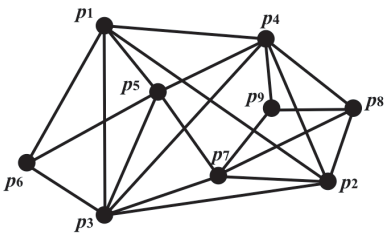


図 2 NSW の例
Fig. 2 Example of NSW.

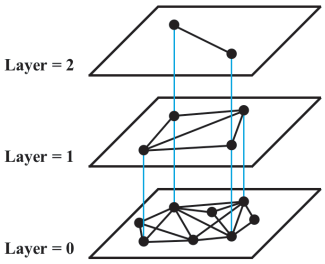


図 3 HNSW の例
Fig. 3 Example of HNSW.

検索結果の k 個のノードとの間に無向エッジを作る．そして，データ追加の操作を繰り返すことで NSW を構築する．NSW の例を**図 2** に示す．NSW は，インデックス構築初期に作成した長いエッジをたどってクエリに近づき，後期に作成した短いエッジをたどってクエリに近いノードを訪問することにより，効率的に検索を行うことが可能である．

その一方で，データを追加する際の順序によってグラフが大きく異なるという問題がある．また，グラフの連結性は保証されていないため，AkNNG と同様，任意のクエリに対して検索を行うことが困難な場合がある．

3.3.3 HNSW：Hierarchical NSW

NSW の問題を解決するため，文献 [25] では，HNSW というグラフインデックスを提案している．HNSW は，複数の NSW を階層的に配置する．さらに，近似 k 最近傍に対してエッジを作るのではなく，検索の過程で得られたあるノードに対してエッジを作り，次にそのノードよりも近いノードを見つけるとそのノードにもエッジを作る．この手順を繰り返してグラフを構築することにより，グラフの連結性を改善することができる．HNSW の例を**図 3** に示す．水色で示した線は，上位層と共通して配置しているノードの関係を表している．

© 2021 Information Processing Society of Japan

HNSW は，実データを用いた実験によって高精度かつ高速な近似 k 最近傍検索ができることが示されている．しかし，複数の NSW を階層的に構築するため，メモリ消費量が大きい．また，NSW と同様，グラフの連結性は保証されていない．

3.4 NSG：Navigating Spreading-out Graph

文献 [11] では，NSG と呼ばれるグラフインデックスを提案しており，ユークリッド空間における近似 k 最近傍検索は，インデックス構築時間，検索の実行時間，およびメモリ消費量の観点で NSG が最高性能であるとされている．Algorithm 1 より，グラフによる検索アルゴリズムの実行時間は，主に検索時にアクセスするノード数で決まることが分かる．NSG は，検索パスを短く，出次数を小さくすることで検索時にアクセスするノード数を削減し，高速な近似 k 最近傍検索を実現する近接グラフである．また NSG では，グラフの連結性が保証されている．以下では，NSG を構成する要素について説明する．

3.4.1 Navigating Node

Navigating Node は，データ空間の重心に最も近い点である．検索を行うときは，任意のクエリに対して Navigating Node を始点とすることにより，平均的に検索時のパスの長さが最短となる．

3.4.2 MRNG

検索パスを短くするためのグラフである MRNG の定義を以下に示す．

定義 4（MRNG：Monotonic Relative Neighbor Graph）．d 次元ユークリッド空間 E^d で定義されたグラフを G，G のノード集合を V，エッジ集合を E，$\overrightarrow{uv} \in E$ を任意のノード $u, v \in V$ 間のエッジとする．G が以下を満たすとき，G は MRNG であるという．

$$lune_{uv} \cap V = \emptyset \text{ or } \overrightarrow{uw} \notin E,\ \forall w \in (lune_{uv} \cap V)$$

ただし，$lune_{uv}$ は u を中心とする半径 $l = distance(u, v)$ の超球と v を中心とする半径 l の超球との共通部分である．**図 4** に MRNG の例を示す．青色の弧で囲まれた領域は $lune$ を表す．MRNG について以下の定理が成り立つ．

定理 1 [11]. d 次元ユークリッド空間 E^d で定義された MRNG を G とし，u，v を G に含まれる任意のノードとする．u を始点として Algorithm 1 を実行することにより，一度も v から遠ざかることなく v を訪問することができる．

したがって，MRNG を構築することにより検索パスを短くできる．厳密な MRNG を構築するには，各ノード v からデータ集合 P の全ノードに対して条件を満たすようにエッジを作る．しかし，それには $O(N^2)$ の時間計算量が必要であるため，NSG では v からデータ集合の部分集合 $Ps \subset P$ のノードに対してエッジを作る．Ps は Navigating Node を始点，v をクエリとして Algorithm 1，その過程で訪問したノードとする．

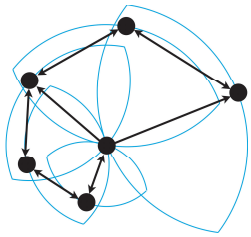


図 4 MRNG
Fig. 4 MRNG.

Algorithm 2: BuildNSG

Input: AkNNG G, max out-degree m and candidate set size l
Output: NSG

```
1  n ← FindNavigatingNode
2  for ∀v ∈ G do
3      candidates of neighbor D ← KNNSearch(v, 1, n, l)
         ∪ {all nodes checked along the KNNSearch}
4      sort D in ascending order of the distance to v
5      p_0 ← the closest node to v in D
6      v.neighbors ← {p_0}
7      for ∀u ∈ D do
8          if Conflict then
9              continue
10         v.neighbors ← v.neighbors ∪ {u}
11         if |v.neighbors| ≥ m then
12             break
13 build NSG with nodes and edges
14 while true do
15     DFS from root n
16     if not all the nodes linked to n then
17         d ← unlinked node
18         w ← KNNSearch(d, 1, n, l)
19         d.neighbors ← d.neighbors ∪ {w}
20     else
21         break
22 return NSG
```

また，出次数を小さくするために，各ノードの最大出次数をハイパーパラメータで制限する．これらにより，Algorithm 1 によって効率的に検索を行うことができる．

3.4.3 NSG 構築アルゴリズム

NSG を構築するアルゴリズムを Algorithm 2 に示す．入力として，事前に構築した AkNNG G，各点の最大出次数 m，および kNN 検索を行うときの候補集合の最大要素数 l を与える．まず，Navigating Node n を求めるため，ランダムにノードを選び，データ集合の重心をクエリとして，Algorithm 1 によって重心の近似最近傍を求める（Algorithm 3）．次に，データ集合の各ノード v に対して以

© 2021 Information Processing Society of Japan

Algorithm 3: FindNavigatingNode

Input: candidate set size l, kNN graph G
Output: Navigating Node

```
r ← random node in G
c ← centroid of nodes in G
return KNNSearch(c, 1, r, l)
```

Algorithm 4: Conflict

Input: process node v, neighbor candidate u and neighbor candidate set D
Output: Boolean

```
if lune_uv ∩ D = ∅ then
    return false
for ∀w ∈ {lune_uv ∩ D} do
    if w ∈ v.neighbors then
        return true
return false
```

下の操作を行う．v をクエリ，n を始点として，Algorithm 1 を実行する．検索の過程で訪問したノードに，MRNG の条件を満たすようエッジを作る（Algorithm 4）．ただし，v の出次数は m に制限する．以上の操作が完了した後，n を根として深さ優先探索を行う．非連結なノード w があった場合は，w をクエリ，n を始点として Algorithm 1 を実行し，検索結果として得られた近似最近傍から w に対してエッジを作る．

Navigating Node は重心を用いて定義されている．しかし，たとえばデータ集合が文字列集合，距離関数が編集距離である場合，データをベクトルとして表現できない．このような場合は重心が定義できないため，メトリック空間における Navigating Node は自明でない．また，MRNG は *lune* を用いて定義されている．*lune* は 2 つの超球の共通部分として定義されているが，超球の共通部分は d 次元実ベクトル空間 $\mathbb{R}^d$，すなわちデータを座標空間にプロットできるような空間における概念であるため，メトリック空間において一般的に定義できない．ゆえに，メトリック空間では，Algorithm 4 を用いて MRNG を構築する際に，1 行目や 4 行目のように *lune* を用いた処理ができない．さらに，グラフインデックスを用いて高速に範囲検索を行うアルゴリズムについてもこれまでに提案されていない．

4. 提案アルゴリズム

提案アルゴリズムでは，メトリック空間に拡張した NSG を用いて近似範囲検索を行う．以下では，NSG のメトリック空間への拡張とグラフを用いた範囲検索アルゴリズムについて述べる．

4.1 NSG のメトリック空間への拡張

4.1.1 メトリック空間における Navigating Node

メトリック空間では重心が定義できないため，Navigating Node をメトリック空間に拡張する必要がある．Navigating Node は，与えられたデータ集合のうち，すべてのデータにとって近いデータが望ましい．したがって，以下のようにメトリック空間における Navigating Node を定義する．

定義 5（Navigating Node）． グラフ G における Navigating Node n は以下を満たす．

$$n = \underset{u \in P}{\operatorname{argmin}} \left(\sum_{v \in P} distance(u, v) \right) \tag{1}$$

4.1.2 メトリック空間における MRNG

定義 4 で述べたように，MRNG は，$lune_{uv}$（ただし $u, v \in V$）を用いて定義されている．しかし $lune_{uv}$ はメトリック空間で利用できない．ここで，u，v からの距離が $l = distance(u, v)$ 未満であるノードの部分集合 S_{uv} を式 (2) のように定義する．

$$S_{uv} = \{w \in V \mid distance(u, w) < l \text{ and } distance(w, v) < l\} \tag{2}$$

S_{uv} は定義に超球を用いていないため，メトリック空間で利用できる．これを用いて，メトリック空間における MRNG を以下のように定義する．

定義 6（メトリック空間における MRNG）． メトリック空間 $\mathcal{M}$ で定義されたグラフを G，G のノード集合を V，エッジ集合を E，$\overrightarrow{uv} \in E$ を任意のエッジとする．G が以下を満たすとき，G は MRNG である．

$$S_{uv} = \emptyset \text{ or } \overrightarrow{uw} \notin E,\ \forall w \in S_{uv} \tag{3}$$

式 (2) より，ユークリッド空間では明らかに $S_{uv} = lune_{uv} \cap V$ であるため，定義 6 は定義 4 のメトリック空間への拡張となっている．

4.2 メトリック空間における NSG の構築アルゴリズム

3.4.3 項で述べた NSG を構築するアルゴリズムのうち，ユークリッド空間に依存する Algorithm 3 および Algorithm 4 を変更する．すなわち，定義 5 に従って Navigating Node を計算する．ただし，式 (1) の計算量は $O(N^2)$ であるため，実際のアルゴリズムでは P からサンプリングしたデータの集合に対して式 (1) を実行し，近似的に Navigating Node を求める．そして，追加するエッジが MRNG の条件を満たすかどうかを Algorithm 5 により調べる．Algorithm 5 では，定義 6 に従って，処理するノード v と隣接ノードの候補である u が与えられたとき，u，v が式 (3) を満たすかどうか判定する．Algorithm 4 と 5 の違いは，定義 4 ではなく定義 6 にしたがって MRNG かどうかを判定

© 2021 Information Processing Society of Japan

Algorithm 5: ConflictInM

Input: process node v and neighbor candidate u
Output: Boolean

```
for ∀w ∈ v.neighbors do
    if u ∉ w.neighbors then
        continue
    if distance(v, w) < distance(v, u) and distance(w,
     u) < distance(v, u)) then
        return true
return false
```

Algorithm 6: NaiveRangeSearch

Input: query q, search range r, start node s, initial result size k_{init} and inclease of k *step*
Output: data in search range

```
result set R ← ∅
k ← k_init
while true do
    candidate set C ← KNNSearch(q, k, s, k)
    e ← the last node of C
    if distance(q, e) < r then
        k ← k ∗ step
    else
        R ← data in C whose distance from q within r
        return R
```

している点である．

以上のように変更した後，Algorithm 2 を実行することで，メトリック空間における NSG を構築できる．

4.3 範囲検索アルゴリズム

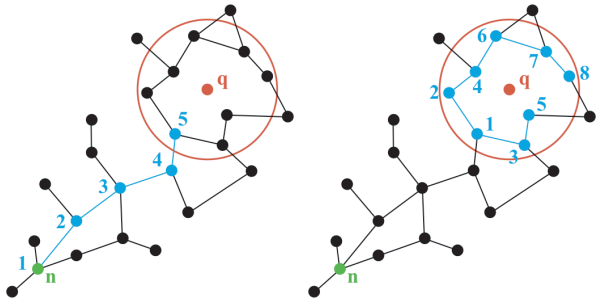
4.3.1 単純なアルゴリズム

グラフインデックスを用いた近似範囲検索アルゴリズムは，k を大きくしながら近似 k 最近傍検索を繰り返すことにより実現できる．その疑似コードを Algorithm 6 に示す．Algorithm 6 では，まず小さい k で近似 k 最近傍検索を行う．得られた近似 k 最近傍とクエリとの距離が検索範囲より小さければ，k を大きくして近似 k 最近傍を再計算する．得られた近似 k 最近傍とクエリとの距離が検索範囲より大きければ，候補集合のうち検索範囲内にあるデータを返却して終了する．

しかし，Algorithm 6 では，検索結果の数によっては何度も k 最近傍検索を繰り返すおそれがあり，そのような場合は低速となる．そのため，効率的に検索範囲内のデータを検索するアルゴリズムが必要となる．

4.3.2 効率的なアルゴリズム

グラフを用いた検索アルゴリズムでは，実行時間を小さくするためには検索パスを短くする必要がある．さらに範囲検索では，検索範囲外のノードに対する訪問回数を可能な限り減らさなければならない．したがって，始点から検索範囲内のノードを発見するまでは，Algorithm 1 を用いてクエリに最も接近するエッジを貪欲にたどることによって検索パスを可能な限り短くするとともに，検索範囲外のノードの訪問回数を減らす．また，検索範囲内においては，検索範囲内のエッジを貪欲にたどることにより，検索範囲外のノードの訪問を避けられる．NSG は，各ノードから各ノードに対して一度も遠ざかることなく近付くことができるので，検索範囲の外に出て再び検索範囲内に入るパスを考慮しなくても高精度な範囲検索が実現できる．



(a) Global Range Search (b) Local Range Search

図 5 グラフによる範囲検索アルゴリズム
Fig. 5 Range search algorithm with graph-based index.

提案アルゴリズムは 2 段階で構成される．k 最近傍検索と同様にエッジをたどって検索範囲内のノードを 1 つ見つけるステップと，検索範囲内のノードを貪欲に訪問するステップである．以後，前者を Global Range Search，後者を Local Range Search と呼ぶ．擬似コードを Algorithm 7 に示す．

Global Range Search は，Algorithm 1 で示した k 最近傍検索と同じ手順で候補集合を更新する．その過程で，検索範囲内のノードを 1 つ見つけると終了し，Local Range Search に移る．もし候補集合が更新されなくなるまで解を見つけることができなければ，解なしとして終了する．Global Range Search において，Navigating Node から後戻りの起こらないパスを利用してクエリに近付く様子を**図 5** (a) に示す．図 5 (a) において，緑色で示したノードが Navigating Node，赤色で示した点がクエリ，赤色の円が検索範囲である．水色で示したノードが Global Range Search の過程で訪問したノードであり，その添字はたどった順番を表す．

Local Range Search は，Global Range Search で見つけた解からそのエッジをたどってノードを訪問する．訪問したノードが解であれば解集合に追加し，さらにその隣接ノードを訪問する．そうでなければそのノードは以後考慮しない．解集合が更新されなくなると Local Range Search を終了し，解集合を返却する．Local Range Search の様子を図 5 (b) に示す．図 5 (b) において，水色で示したノー

© 2021 Information Processing Society of Japan

Algorithm 7: RangeSearch

Input: query q, search range r, start node s and candidate set size l
Output: data in search range

```
result set R ← ∅
// Global Range Search
candidate set C ← {s}
while true do
    i ← the index of the first unvisited node in C
    if i ≥ l then
        break
    v ← i-th candidate in C
    mark v as visited
    for ∀u ∈ p.neighbors do
        C ← C ∪ {u}
        if distance(q, u) < r then
            R ← {u}
            break
    if |R| > 0 then
        break
    sort C in ascending order of the distance to q
    if |C| > l then
        C.resize(l)
if |R| = ∅ then
    return ∅
// Local Range Search
while true do
    i ← the index of the first unvisited node in R
    if checked all nodes in R then
        return R
    v ← i-th result in R
    mark v as visited
    for ∀u ∈ v.neighbors do
        if distance(q, u) < r then
            R ← R ∪ {u}
```

ドが Local Range Search の過程で訪問したノードであり，その添字はたどった順番を表す．

5. 評価実験

本章では，提案アルゴリズムの性能評価のために行った実験の結果を示す．

5.1 設定

本実験では以下の 8 つのアルゴリズムを評価した．

- メトリック空間に拡張した NSG（M-NSG と表記する）上で実行した範囲検索アルゴリズム．
- NSG 上で実行した範囲検索アルゴリズム．
- Algorithm 3 で Navigating Node を計算した M-NSG 上で実行した範囲検索アルゴリズム．
- M-NSG 上で実行した Algorithm 6．
- HNSW [25]．nmslib [*1] による実装を利用した．
- 線形探索（Scan と表記する）．faiss [*2] による実装を利用した．
- VP 木 [32]．
- LSH．l_p 距離に対しては文献 [7]，角距離に対しては文献 [2] におけるアルゴリズムを評価した．

以上のアルゴリズムは C++で実装されており，すべての実験は 3.0 GHz Intel Core i9 および 128 GB RAM で構成される PC 上で行った．また，すべてのアルゴリズムはシングルスレッドにより評価した．現在，LSH [7] から派生した様々なアルゴリズムが提案されているが，それらは近似 k 最近傍検索のために設計されているため，そのまま近似範囲検索に適用することはできない．したがって，比較手法としての LSH として，文献 [7] および文献 [2] が提案しているアルゴリズムを選択した．LSH のパラメータに関しては，実行時間と精度のトレードオフがとれたところでチューニングを行った．提案手法 M-NSG の事前インデックスである AkNNG は，文献 [8] で提案しているアルゴリズムを用いて構築した．

5.1.1 評価指標

本実験では，10,000 個のクエリを発行したときの以下の 2 つの指標を評価した．

- 実行時間：1 クエリあたりの処理時間の中央値．
- Recall：検索結果の Recall の中央値．近似範囲検索アルゴリズムの特性上，False Negative が返却されることがないため，精度の評価指標として Recall が最適である．

SIFT を用いた実験では，データセットに含まれている 10,000 個のクエリを用いた．GloVe はクエリのためのデータセットが提供されていないため，データの中からランダムに 10,000 個サンプリングしたものをクエリとした．HEPMASS は，クエリのデータセットの中からランダムに 10,000 個サンプリングしたものを使用した．

5.1.2 データセット

本実験では，3 つの実データ SIFT [*3]，GloVe [*4]，および HEPMASS [*5] を用いた．SIFT は画像の特徴量である SIFT の特徴量記述子を集めたデータセットである．GloVe は Twitter で学習した自然言語の単語の埋め込み表現である．HEPMASS は，素粒子の衝突シミュレーションのデータセットである．**表 1** にデータセットの情報と利用する距離関数を示す．

[*1] https://github.com/nmslib/hnswlib
[*2] https://github.com/facebookresearch/faiss
[*3] http://corpus-texmex.irisa.fr/
[*4] https://nlp.stanford.edu/projects/glove/
[*5] https://archive.ics.uci.edu/ml/datasets/HEPMASS

© 2021 Information Processing Society of Japan

表 1 データセット
Table 1 Dataset.

データセット	データ数	d	距離関数
SIFT	1,000,000	128	ユークリッド距離
GloVe	1,193,514	25	角距離
HEPMASS	7,000,000	28	マンハッタン距離

表 2 パラメータ
Table 2 Parameters.

データセット	パラメータ	値
SIFT	r	175, 200, 225, **250**, 275
	N	200,000, 400,000, 600,000, 800,000, **1,000,000**
	k	**50**
	l	**50**
	m	**50**
	N_s	**10,000**
GloVe	r	0.15, 0.17, 0.19, **0.21**, 0.23
	N	200,000, 400,000, 600,000, 800,000, **1,000,000**
	k	**40**
	l	**40**
	m	**40**
	N_s	**10,000**
HEPMASS	r	10, 11, 12, **13**, 14
	N	1,000,000, 1,500,000, **2,000,000**, 2,500,000, 3,000,000
	k	**40**
	l	**120**
	m	**60**
	N_s	**10,000**

5.1.3 パラメータ

以下にインデックス構築および範囲検索アルゴリズムのパラメータを示す．

- r：検索範囲
- N：検索対象のデータ数
- k：事前インデックスとして用いる AkNNG の出次数
- l：NSG 構築および検索時の候補集合の最大要素数
- m：NSG の最大出次数
- N_s：Navigating Node を求める際のサンプル数

表 2 にパラメータの設定を示す．太字で示している数字がデフォルトのパラメータである．提案手法のシステムパラメータである k，l，m，N_s については，実行時間と精度のトレードオフが取れたところでチューニングを行った．

5.2 実験結果

5.2.1 NSG との比較

NSG のメトリック空間の拡張による Recall への影響を検証するため，ユークリッド空間における NSG を用いて提案アルゴリズムを実行したときとの比較を行った．また，

表 3 M-NSG と NSG の Recall の比較
Table 3 Comparison of recall between M-NSG and NSG.

	Recall
M-NSG	0.98
NSG	0.98

表 4 Navigating Node のメトリック空間への拡張の Recall への影響
Table 4 Influence of extention to metric space.

	SIFT	GloVe	HEPMASS
提案手法	0.98	0.96	0.98
Algorithm 3	0.98	0.96	0.98

表 5 サンプリングによる影響
Table 5 Influence of sampling.

		SIFT	GloVe	HEPMASS
w/ sampling	構築時間 [min]	66	56	84
	Recall	0.98	0.96	0.98
w/o sampling	構築時間 [min]	671	526	1189
	Recall	0.99	0.96	0.97

ユークリッド空間における両者の比較を行うため，データセットは SIFT を用いた．その結果を**表 3** に示す．

表 3 より，SIFT データセットとユークリッド距離を用いて近似範囲検索を行ったとき，Recall に大きな差は見られない．このことから，提案手法は性能を低下させることなくメトリック空間に拡張できていることが分かる[*6]．

5.2.2 Navigating Node のメトリック空間への拡張による影響

Navigating Node のメトリック空間への拡張による影響を調べるため，Algorithm 3 により計算した Navigating Node と提案手法により計算した Navigating Node を用いたときの Recall の比較を行った．その結果を**表 4** に示す．

表 4 より，メトリック空間への拡張による Recall への影響はほとんどない．したがって，提案手法により，性能を落とすことなくメトリック空間に拡張できていることが分かる．

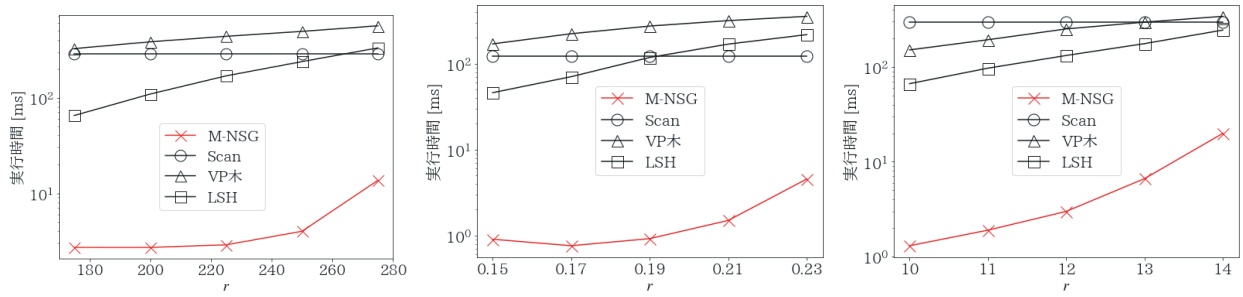
5.2.3 Navigating Node を計算する際のサンプリングによる影響

4.2 節で説明したように，提案手法ではメトリック空間に拡張した Navigating Node の計算を高速化するためにサンプリングを行っている．このサンプリングによるインデックス構築時間および Recall への影響を調べた結果を**表 5** に示す．

表 5 より，サンプリングは，Recall に影響を与えずに高速にインデックスを構築できることが分かる．

[*6] ユークリッド空間における NSG を GloVe と HEPMASS に用いた結果，Recall はそれぞれ 0.86，0.03 となり Recall が大幅に低下した．

© 2021 Information Processing Society of Japan



(a) SIFT (b) GloVe (c) HEPMASS

図 6 r の影響（実行時間）
Fig. 6 Influence of r (search time).

表 6 Algorithm 6 と 7 との比較
Table 6 Comparison between naive and proposed method.

		SIFT	GloVe	HEPMASS
Algorithm 6	実行時間 [ms]	10.54	510.70	583.86
	Recall	1.00	1.00	1.00
Algorithm 7	実行時間 [ms]	4.00	1.50	6.63
	Recall	0.98	0.96	0.98

表 7 M-NSG と HNSW との比較
Table 7 Comparison between HNSW and proposed method.

		SIFT	GloVe
M-NSG	実行時間 [ms]	4.00	1.50
	Recall	0.98	0.96
HNSW	実行時間 [ms]	18.53	13.85
	Recall	1.00	1.00

5.2.4 単純なアルゴリズムとの比較

提案手法である Algorithm 7 の有効性を示すため，Algorithm 6 との比較結果を**表 6** に示す．パラメータは，$k_{init} = 100$，$step = 100$ とした．

表 6 より，Algorithm 6 は，検索範囲が大きくなることが多いため，Recall は 1.00 となっているものの低速である．また，アプリケーションが近似範囲検索を行うときは，Recall が 1 であることを要求せず，高速性を求めている．一方で，Algorithm 7 は 1 に近い Recall を保ちながら，Algorithm 6 と比較して最大約 340 倍の高速化を達成している．

5.2.5 HNSW との比較

HNSW を用いた近似 k 最近傍検索アルゴリズムとの比較を行った結果を**表 7** に示す．近似 k 最近傍検索アルゴリズムで近似範囲検索を行うため，k を検索結果の数に設定した．また，nmslib の HNSW はマンハッタン距離に対応していなかったため，HEPMASS データセットを用いた評価は行わない．

表 7 より，提案手法は 1 に近い Recall を保ったまま，HNSW よりも 4 倍から 8 倍の高速化を達成している．

5.2.6 パラメータの影響

最後に，既存の範囲検索アルゴリズムと提案アルゴリズムを r と N を変化させながら比較する．

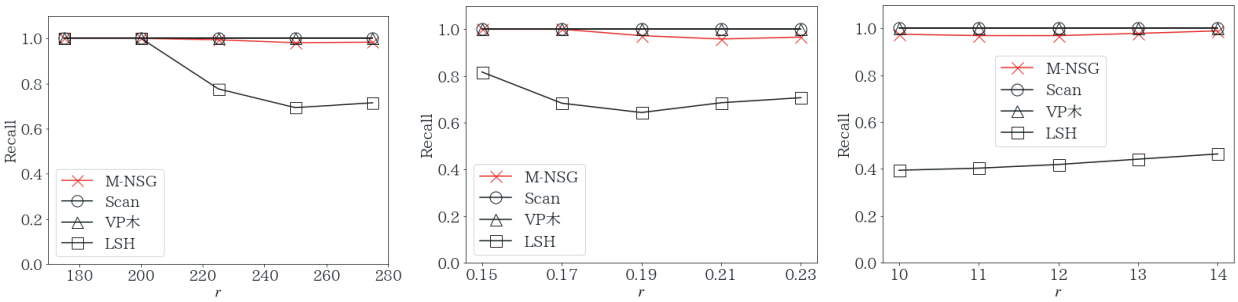
r の影響． r を変化させたときの実験結果を**図 6**，および**図 7** に示す．

Scan 以外のアルゴリズムは，解の数が多ければ多いほど訪問するノードの数が多くなるため，r が大きくなるほど実行時間が大きくなる．VP 木を用いたアルゴリズムは，Scan と同程度の実行時間がかかっている．これは，次元の呪いによって三角不等式を用いた枝刈りができていないことが原因である．それに対し，提案手法は Scan および VP 木と比較して 100 倍から 200 倍高速に動作している．提案手法では，Global Range Search において，クエリに接近する隣接ノードのみを訪問し，Local Range Search では，検索範囲内のノードを貪欲に訪問する．すなわち，提案手法では距離の差を用いて枝刈りを行うのではなく，クエリに接近しないパスの枝刈りを行う．そして M-NSG は，その過程で遠回りが起こらないようにパスが構築されているため，VP 木に対して，検索範囲外のノードに対する無駄なアクセスが減少している．さらに提案手法は，LSH と比較して 30 倍から 100 倍高速に動作しており，Recall も安定して高い．LSH は，互いに近いデータどうしが同じバケットに格納されやすいという性質を持つハッシュ関数を利用しているが，その確率はそれほど高くない．そのため，クエリと同じバケットの中に，解となりうるデータが存在せず，クエリとの距離が大きいデータが格納される事象が頻繁に発生する．それに対して M-NSG は，距離を正確に計算するため，自身に近いノードにエッジを作れる．これにより，検索の過程で，クエリに近いノードを取りこぼしたり，意図せずクエリに遠いノードにアクセスする可能性が低くなる．

N の影響． N を変化させたときの実験結果を**図 8**，および**図 9** に示す．

グラフを用いた提案アルゴリズムは，検索対象のデータ量が変化しても，Scan および VP 木と比較して 100 倍から 200 倍，LSH と比較して 30 倍から 100 倍高速に動作して

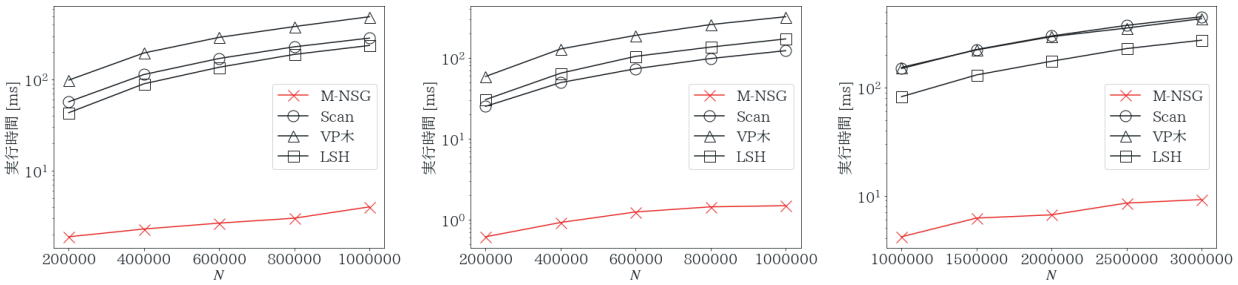
© 2021 Information Processing Society of Japan



(a) SIFT (b) GloVe (c) HEPMASS

図 7 r の影響（Recall）

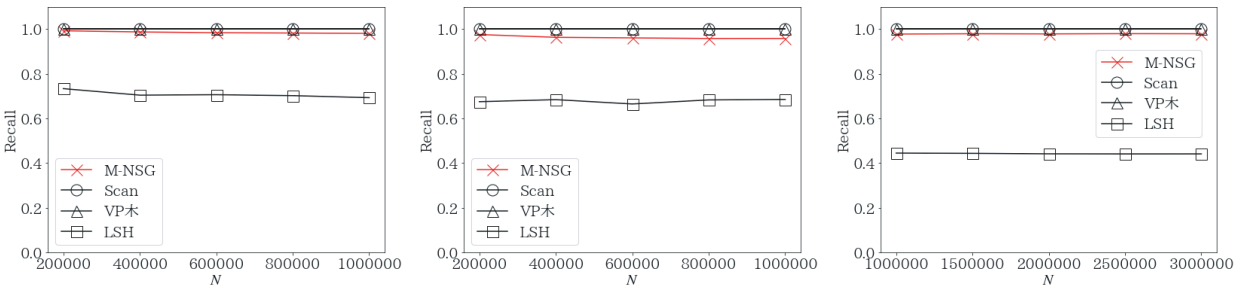
Fig. 7 Influence of r (Recall).



(a) SIFT (b) GloVe (c) HEPMASS

図 8 N の影響（実行時間）

Fig. 8 Influence of N (search time).



(a) SIFT (b) GloVe (c) HEPMASS

図 9 N の影響（Recall）

Fig. 9 Influence of N (Recall).

いる．また，他の手法と比較して，データ量が増加したときの実行時間の増加量が緩やかであり，スケーラビリティが高い．

6. 結論

本論文では，ユークリッド空間において高速に近似 k 最近傍検索を実行するデータ構造である NSG をメトリック空間に拡張するとともに，グラフを用いて効率的に近似範囲検索を行うアルゴリズムを提案した．提案アルゴリズムは，後戻りすることなくエッジをたどることで検索範囲内のノードを発見する Global Range Search と，検索範囲内のエッジを貪欲にたどることで検索範囲内のノードを訪問する Local Range Search で構成される．提案アルゴリズムにより，検索範囲外のノードの訪問を減らすことができるため，高速に近似範囲検索を実行することができる．

実データを用いた実験の結果から，提案アルゴリズムは，木構造やハッシュによる範囲検索と比較して高速に動作することを確認した．

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），JST さきがけ（JPMJPR1931），および JST CREST（J181401085）の支援を受けたものである．

参考文献

[1] Andoni, A. and Indyk, P.: Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions, *Proc. Annual IEEE Symposium on Foundations of Computer Science*, pp.459–468 (online), DOI: 10.1109/FOCS.2006.49 (2006).

© 2021 Information Processing Society of Japan

[2] Andoni, A., Indyk, P., Laarhoven, T., Razenshteyn, I.P. and Schmidt, L.: Practical and Optimal LSH for Angular Distance, *Annual Conf. Neural Information Processing Systems 2015*, pp.1225–1233 (2015) (online), available from ⟨http://papers.nips.cc/paper/5893-practical-and-optimal-lsh-for-angular-distance⟩.

[3] Belkin, N.J. and Croft, W.B.: Information Filtering and Information Retrieval: Two Sides of the Same Coin?, *Comm. ACM*, Vol.35, No.12, pp.29–38 (1992) (online), DOI: 10.1145/138859.138861.

[4] Bentley, J.L.: Multidimensional Binary Search Trees Used for Associative Searching, *Comm. ACM*, Vol.18, No.9, pp.509–517 (online), DOI: 10.1145/361002.361007 (1975).

[5] Chen, L., Gao, Y., Zheng, B., Jensen, C.S., Yang, H. and Yang, K.: Pivot-based Metric Indexing, *Proc. Very Large Data Bases*, Vol.10, No.10, pp.1058–1069 (online), DOI: 10.14778/3115404.3115411 (2017).

[6] Comer, D.: The Ubiquitous B-Tree, *ACM Computing Surveys*, Vol.11, No.2, pp.121–137 (online), DOI: 10.1145/356770.356776 (1979).

[7] Datar, M., Immorlica, N., Indyk, P. and Mirrokni, V.S.: Locality-sensitive hashing scheme based on p-stable distributions, *Proc. ACM Symposium on Computational Geometry*, pp.253–262 (online), DOI: 10.1145/997817.997857 (2004).

[8] Dong, W., Charikar, M. and Li, K.: Efficient k-nearest neighbor graph construction for generic similarity measures, *Proc. Int'l Conf. World Wide Web*, pp.577–586 (online), DOI: 10.1145/1963405.1963487 (2011).

[9] Ester, M., Kriegel, H., Sander, J. and Xu, X.: A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise, *Proc. Int'l Conf. Knowledge Discovery and Data Mining*, pp.226–231 (online), available from ⟨http://www.aaai.org/Library/KDD/1996/kdd96-037.php⟩ (1996).

[10] Fu, C. and Cai, D.: EFANNA: An Extremely Fast Approximate Nearest Neighbor Search Algorithm Based on kNN Graph, *Computing Research Repository*, Vol.abs/1609.07228 (2016) (online), available from ⟨http://arxiv.org/abs/1609.07228⟩.

[11] Fu, C., Xiang, C., Wang, C. and Cai, D.: Fast Approximate Nearest Neighbor Search With The Navigating Spreading-out Graphs, *Proc. Very Large Data Bases*, Vol.12, No.5, pp.461–474 (2019).

[12] Ge, T., He, K., Ke, Q. and Sun, J.: Optimized Product Quantization for Approximate Nearest Neighbor Search, *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, pp.2946–2953 (online), DOI: 10.1109/CVPR.2013.379 (2013).

[13] Gionis, A., Indyk, P. and Motwani, R.: Similarity Search in High Dimensions via Hashing, *Proc. Int'l Conf. Very Large Data Bases*, pp.518–529 (1999) (online), available from ⟨http://www.vldb.org/conf/1999/P49.pdf⟩.

[14] Greg, L., Brent, S. and Jeremy, Y.: Amazon.com Recommendations: Item-to-Item Collaborative Filtering, *IEEE Internet Computing*, Vol.7, pp.76–80 (2003).

[15] Hajebi, K., Abbasi-Yadkori, Y., Shahbazi, H. and Zhang, H.: Fast Approximate Nearest-Neighbor Search with k-Nearest Neighbor Graph, *Proc. Int'l Joint Conf. Artificial Intelligence*, pp.1312–1317 (online), DOI: 10.5591/978-1-57735-516-8/IJCAI11-222 (2011).

[16] Harwood, B. and Drummond, T.: FANNG: Fast Approximate Nearest Neighbour Graphs, *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, pp.5713–5722 (online), DOI: 10.1109/CVPR.2016.616 (2016).

[17] Huang, Q., Feng, J., Zhang, Y., Fang, Q. and Ng, W.: Query-Aware Locality-Sensitive Hashing for Approximate Nearest Neighbor Search, *Proc. Very Large Data Bases*, Vol.9, No.1, pp.1–12 (online), DOI: 10.14778/2850469.2850470 (2015).

[18] Jégou, H., Douze, M. and Schmid, C.: Product Quantization for Nearest Neighbor Search, *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol.33, No.1, pp.117–128 (online), DOI: 10.1109/TPAMI.2010.57 (2011).

[19] Johnson, J., Douze, M. and Jégou, H.: Billion-scale similarity search with GPUs, *Computing Research Repository*, Vol.abs/1702.08734 (online), available from ⟨http://arxiv.org/abs/1702.08734⟩ (2017).

[20] Kalantidis, Y. and Avrithis, Y.: Locally Optimized Product Quantization for Approximate Nearest Neighbor Search, *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, pp.2329–2336 (online), DOI: 10.1109/CVPR.2014.298 (2014).

[21] Li, H., Chan, T.N., Yiu, M.L. and Mamoulis, N.: FEXIPRO: Fast and Exact Inner Product Retrieval in Recommender Systems, *Proc. Int'l Conf. Management of Data*, pp.835–850 (online), DOI: 10.1145/3035918.3064009 (2017).

[22] Liu, Y., Cui, J., Huang, Z., Li, H. and Shen, H.T.: SK-LSH: An Efficient Index Structure for Approximate Nearest Neighbor Search, *Proc. Very Large Data Bases*, Vol.7, No.9, pp.745–756 (online), DOI: 10.14778/2732939.2732947 (2014).

[23] Lv, Q., Josephson, W., Wang, Z., Charikar, M. and Li, K.: Multi-Probe LSH: Efficient Indexing for High-Dimensional Similarity Search, *Proc. Int'l Conf. Very Large Data Bases*, pp.950–961 (online), available from ⟨http://www.vldb.org/conf/2007/papers/research/p950-lv.pdf⟩ (2007).

[24] Malkov, Y., Ponomarenko, A., Logvinov, A. and Krylov, V.: Approximate nearest neighbor algorithm based on navigable small world graphs, *Infomation Systems*, Vol.45, pp.61–68 (online), DOI: 10.1016/j.is.2013.10.006 (2014).

[25] Malkov, Y.A. and Yashunin, D.A.: Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs, *IEEE Trans. Pattern Analysis and Machine Intelligence* (2018).

[26] Rodriguez, A. and Laio, A.: Clustering by fast search and find of density peaks, *Science*, pp.1492–1496 (2014).

[27] Sun, Y., Wang, W., Qin, J., Zhang, Y. and Lin, X.: SRS: Solving c-Approximate Nearest Neighbor Queries in High Dimensional Euclidean Space with a Tiny Index, *Proc. Very Large Data Bases*, Vol.8, No.1, pp.1–12 (online), DOI: 10.14778/2735461.2735462 (2014).

[28] Sural, S., Qian, G. and Pramanik, S.: A comparative analysis of two distance measures in color image databases, *Proc. Int'l Conf. Image Processing*, pp.401–404 (online), DOI: 10.1109/ICIP.2002.1038045 (2002).

[29] Takuma, D. and Yanagisawa, H.: Faster upper bounding of intersection sizes, *Proc. Int'l Conf. Research and Development in Information Retrieval*, pp.703–712 (online), DOI: 10.1145/2484028.2484065 (2013).

[30] Teflioudi, C., Gemulla, R. and Mykytiuk, O.: LEMP: Fast Retrieval of Large Entries in a Matrix Product, *Proc. Int'l Conf. Management of Data*, pp.107–122 (online), DOI: 10.1145/2723372.2747647 (2015).

[31] Weber, R., Schek, H. and Blott, S.: A Quantitative

© 2021 Information Processing Society of Japan

Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces, *Proc. Very Large Data Bases*, Morgan Kaufmann, pp.194–205 (1998) (online), available from ⟨http://www.vldb.org/conf/1998/p194.pdf⟩.
[32] Yianilos, P.N.: Data Structures and Algorithms for Nearest Neighbor Search in General Metric Spaces, *Proc. ACM/SIGACT-SIAM Symposium on Discrete Algorithms*, pp.311–321 (1993) (online), available from ⟨http://dl.acm.org/citation.cfm?id=313559.313789⟩.
[33] Zheng, Y., Guo, Q., Tung, A.K.H. and Wu, S.: LazyLSH: Approximate Nearest Neighbor Search for Multiple Distance Functions with a Single Index, *Proc. Int'l Conf. Management of Data*, pp.2023–2037 (online), DOI: 10.1145/2882903.2882930 (2016).

新井 悠介

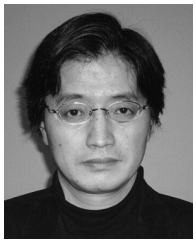
2020 年大阪大学工学部電子情報工学科卒業．同年同大学院情報科学研究科博士前期課程に入学し，現在に至る．

天方 大地 （正会員）

2012 年大阪大学工学部電子情報工学科卒業．2014 年同大学大学院情報科学研究科博士前期課程修了．2015 年同大学院情報科学研究科博士後期課程修了後，同年同大学院情報科学研究科マルチメディア工学専攻助教となり，現在に至る．情報科学博士．2020 年 IEEE Computer Society Japan Chapter Young Author Award 受賞．データベース環境におけるデータ検索技術に関する研究に従事．IEEE，ACM，日本データベース学会各会員．

原 隆浩 （正会員）

1995 年大阪大学工学部情報システム工学科卒業．1997 年同大学大学院工学研究科博士前期課程修了．同年同大学院工学研究科博士後期課程中退後，同大学院工学研究科助手，2004 年同大学院情報科学研究科准教授．2015 年より同大学院情報科学研究科教授となり，現在に至る．工学博士．2003 年本学会研究開発奨励賞受賞 2008 年，2009 年本学会論文賞，2015 年日本学術振興会賞受賞．ネットワーク環境におけるデータ管理技術に関する研究に従事．



藤田 澄男 （正会員）

ヤフー株式会社 Yahoo! JAPAN 研究所特任研究員．1989 年パリ第 7 大学 DEA 修了．Web 検索，各種専門分野検索など多様な情報検索技術とその応用に関する研究・開発に従事．ACM 会員．

© 2021 Information Processing Society of Japan