近接グラフを用いた メトリック空間における効率的な近似範囲検索アルゴリズム

新井　悠介† 天方　大地††,††† 原　隆浩†† 藤田　澄男††††

† 大阪大学工学部　〒 565–0871 大阪府吹田市山田丘 2-1
†† 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　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．近年，画像や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．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，次元圧縮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．本稿では，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，任意の距離関数による検索を可能にする．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　範囲検索，グラフインデックス

1. はじめに

範囲検索は，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．範囲検索は，データベース [3]，情報検索 [24]，情報推薦 [16], [25]，データマイニング [6], [22] など，多くの分野で応用されている．範囲検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全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．そこで，主に木構造を用いた範囲検索アルゴリズムが提案されてきた [2], [26]．しかしながら，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．木構造を用いた正確な範囲検索は，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 [4], [10], [13], [17], [18], [23], [27] や量子化 [9], [15] によって次元を削減する近似手法が提案されている．しかし，このような次元削減手法は利用する距離関数に強く依存するため，メトリック空間における次元削減手法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 [7], [8], [11], [12], [19], [20] が，次元削減を用いる方法よりも精度と速度の両方で優れた性能を示している．

本稿では，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，データ同士の関係をエッジとして表現し，エッジを辿る操作を繰り返して範囲検索を行う．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かつ高精度な近似範囲検索を実現する．

研究動機． メトリック空間における高次元データの高速かつ高精度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，あるユーザが過去に閲覧した商品と類似した商品画像を持つ商品をそのユーザに推薦する状況を考える．このときサーバはユーザが過去に閲覧した商品画像をクエリ q に，距離 r を検索範囲として，扱っている商品画像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．

EC サイトでは低遅延なレスポンスが求められるため，高速なアルゴリズムが必要となる．また，必ずしも正確な結果が求められる訳ではないが，精度が低い近似アルゴリズムを使うと推薦によってユーザの購買を促進できない．さらに，商品画像および EC サイトの特性によっては，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て推薦することによりユーザの嗜好を反映できる可能性があるため [21]，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．

提案アルゴリズムの概要． 本稿では，NSG: Navigating Spreading-out Graph [8] と呼ばれるユークリッド空間における 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，高速にクエリに接近し，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全て検索す

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使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辿ることにより効率的な範囲検索を実現する．

貢献． 以下に本研究の貢献を示す．

（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用性を確認する．

本稿の構成． 以下では，2 章で本稿の問題の定義を述べ，3 章で関連研究を紹介する．4 章で本稿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．最後に 6 章で本稿をまとめる．

2. 予 備 知 識

本問題を定義するために，本稿で用いる用語を定義する．

定義 1（範囲検索）． クエリ q，検索範囲 r，およびデータ集合 $P = \{p_1, p_2, \ldots, p_N\} \subset \mathcal{M}$ が与えられたとする．範囲検索は，P の中で q との距離が r 未満のデータを解として返却する．

問題定義． 本問題は，q，r，および P が与えられたとき，高速かつ高精度に近似範囲検索を実行することである．

3. 関 連 研 究

これまでに多くの高速な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 k 最近傍検索アルゴリズムについて述べる．

3.1 木構造を用いたメトリック空間における範囲検索．

文献 [26] では，VP 木と呼ばれるデータ構造が提案されている．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点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．しかし高次元空間では枝刈り効率が下がるため高速化ができない．

3.2 ハッシュを用いた近似範囲検索．

文献 [10] では，LSH: Locality Sensitive Hashing 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 は距離の小さいデータ同士が高確率で衝突するハッシュ関数である．データ集合の全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 は精度が低く，解となるデータが他のバケットに格納されることが多い．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 [18] 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 [23] 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．

しかしながら，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 Multi-Probe LSH や SRS を利用できない．

文献 [1], [4], [27] 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提案されていない．

3.3 近接グラフを用いた近似 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，高次元空間における近似近傍探索のためのデータ構造として有望視されている [7], [8], [11], [12], [19], [20]．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盛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説明する．

k 最近傍検索の定義を以下に示す．

定義 2（k 最近傍検索）． クエリ q，検索結果の数 k，およびデータ集合 $P = \{p_1, p_2, \ldots, p_N\} \subset \mathcal{M}$ 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 個の点を返却する．

近接グラフを用いた近似 k 最近傍検索アルゴリズムをアルゴリズム 1 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 からノードを取り出し，エッジを辿って訪問したノードを 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．

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 unchecked node in C
    if i ≧ l then
        break
    v ← i-th candidate
    for ∀u ∈ v.neighbors do
        C ← C ∪ {u}
    sort C in ascending order of the distance to q
    if |S| > l then
        C.resize(l)
return top k candidates in C
```

以下では，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 kNN Graph である．kNN Graph 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 [5], [7], [14]．AkNNG をインデックスとして用いると，データが一様に分布しているような空間において，高次元であっても高速に検索を行うことができる．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 [19] 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，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 は，インデックス構築初期に作成した長いエッジを辿ってクエリに近づき，後期に作成した短いエッジを辿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 [20] では，HNSW という近接グラフを提案している．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 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，この手法に対して理論的根拠が与えられていないため，その性能はデータ分布に依存する．また，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．

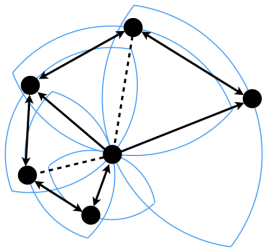


図 1: MRNG

3.4 NSG

文献 [8] では，NSG というグラフインデックスが提案されており，ユークリッド空間における近似 k 最近傍検索は，実行時間とメモリ消費量の両方の観点で NSG が最高性能であるとされている．アルゴリズム 1 より，グラフによる検索アルゴリズムの実行時間は，主に検索時のパスの長さとノードの出次数で決まることがわかる．NSG は，検索パスを短く，出次数を小さくすることで高速な近似 k 最近傍検索を実現する近接グラフである．さらに 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

文献 [8] では，検索パスを短くするためのグラフとして，MRNG を提案している．MRNG の定義を以下に示す．

定義 3（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，$\overrightarrow{uv}$ を G に含まれる任意のエッジとする．G が以下を満たすとき，G は MRNG であるという．

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

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 1 に MRNG の例を示す．

MRNG について以下の定理が成り立つ．

定理 1 [8]． d 次元ユークリッド空間 E^d で定義された MRNG を G とする．アルゴリズム 1 により，G に含まれる任意のノード p，q 間を一度も遠ざかることなく見つけることができる．

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

Algorithm 2: BuildNSG

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

```
n ← FindNavigatingNode
for ∀v ∈ G do
    candidates of neighbor D ← KNNSearch(v, 1, n, l) ∪ {all nodes checked along the KNNSearch}
    sort D ascending order of the distance to v
    p_0 ← the closest node to v in D
    v.neighbors ← {p_0}
    for ∀u ∈ D do
        if Conflict then
            continue
        v.neighbors ← v.neighbors ∪ {u}
        if |v.neighbors| ≧ m then
            break
build NSG with nodes and edges
while true do
    DFS from root n
    if not all the nodes linked to n then
        d ← unlinked node
        w ← KNNSearch(d, 1, n, l)
        d.neighbors ← d.neighbors ∪ {w}
    else
        break
return NSG
```

してエッジを作る．Ps は Navigating Node を始点，p をクエリとしてアルゴリズム 1，その過程で訪問したノードとする．

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，アルゴリズム 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アルゴリズム 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，アルゴリズム 1 によって重心の近似最近傍を求める（アルゴリズム 3）．次に，データ集合の各ノード v に対して以下の操作を行う．v をクエリ，n を始点として，アルゴリズム 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（アルゴリズム 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，n を始点としてアルゴリズム 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 や MRNG など NSG の核となる要素はユークリッド空間の性質を利用しているため，メトリック空間で利用できない．また，範囲検索を効率的に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．

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
```

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．以下にメトリック空間における Navigating Node を定義する．

定義 4（Navigating Node）．

$$Navigating\ Node = \underset{\forall p \in P}{\operatorname{argmin}} \left(\sum_{\forall q \in P} distance(p, q) \right)$$

4.1.2 メトリック空間における MRNG

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 5（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 とし，$\overrightarrow{uv}$ を G に含まれる任意のエッジとする．G が以下を満たすとき，G は MRNG であるという．

$$\overrightarrow{uw} \notin G,\ \forall w \in G,$$
$$distance(u, w) < distance(w, v)\ and$$
$$distance(w, v) < distance(u, v)\ or$$
$$\overrightarrow{uw}, \overrightarrow{wv} \in G,\ \forall w \in G,$$
$$distance(u, w) > distance(u, v)\ or$$
$$distance(w, v) > distance(u, v)$$

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

Algorithm 5: ConflictInM

Input: process node v, 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
```

クリッド空間に依存するアルゴリズム 3 およびアルゴリズム 4 を変更する．すなわち，定義 4 に従って Navigating Node を計算する．ただし，定義 4 では $O(N^2)$ の時間計算量がかかるため，実際のアルゴリズムでは近似を行い，P からサンプリングした部分集合 P_s の中から Navigating Node を選ぶ．

そして，追加するエッジが MRNG の条件を満たすかどうかをアルゴリズム 5 により調べる．アルゴリズム 5 では，定義 5 にしたがって，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，エッジを作るかどうかを判定する．すなわち，v の近傍 w の近傍に u があるかどうかを調べ，u, v, w が条件 $distance(v, w) < distance(v, u)$ かつ $distance(w, u) < distance(v, u)$ を満たすかどうか判定する．条件を満たす場合は真を返し，それ以外の場合は偽を返す．

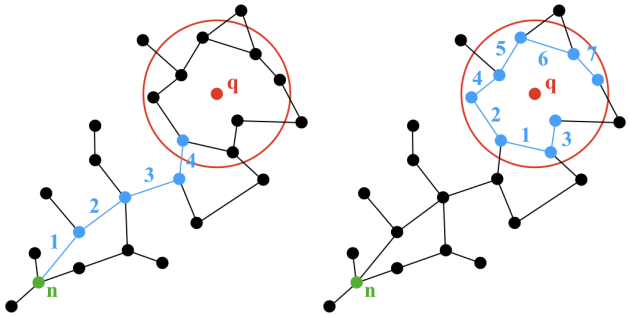
以上のように変更した後，アルゴリズム 2 を実行することで，メトリック空間における NSG を構築できる．

4.3 範囲検索アルゴリズム

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，アルゴリズム 1 を用いてクエリに最も接近するエッジを貪欲に辿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辿ることにより，検索範囲外のノードの訪問を避けられる．NSG は，各点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と考えられる．

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辿って検索範囲内のノードを 1 つ見つけるステップと，検索範囲内で全ての解を求めるステップである．以後，前者を Global Range Search，後者を Local Range Search と呼ぶ．擬似コードをアルゴリズム 6 に示す．

Global Range Search は，アルゴリズム 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 monotonic path を利用して



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

図 2: グラフによる範囲検索アルゴリズム

クエリに近付く様子を図 2a に示す．図 2a において，緑色で示したノードが Navigating Node，赤色で示したノードがクエリ，赤色の円が検索範囲である．水色で示したエッジが Global Range Search の過程で辿ったエッジであり，その添字は辿った順番を表す．

Local Range Search は，Global Range Search で見つけた解からそのエッジを辿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 2b に示す．図 2b において，水色で示したエッジが Local Range Search の過程で辿ったエッジであり，その添字は辿った順番を表す．

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 4 つのアルゴリズムを評価した．

- 本稿の提案アルゴリズム（M-NSG と表記する）．
- 線形探索（Scan と表記する）．
- [26] における提案アルゴリズム（VP 木と表記する）．
- [4] における提案アルゴリズム（LSH と表記する）．ユークリッド距離を用いた実験に使用する．
- NSG の事前インデックスとして用いた AkNNG．メトリック空間における Navigating Node および MRNG の有効性を検証するため，範囲検索アルゴリズムはアルゴリズム 6 を利用し，Global Range Search の始点はランダムに決定する．また，AkNNG は，[5] で提案されている手法により構築する．

以上のアルゴリズムは C++で実装されており，全ての実験は 3.0GHz Intel Core i9 および 128GB RAM で構成される PC 上で行った．

評価指標． 本実験では，以下の 2 つの指標を評価した．

- 実行時間：クエリ処理時間の中央値．
- Recall：検索結果の Recall の中央値．

データセット． 本実験では，2 つの実データ SIFT(注1) および GloVe(注2) を用いた．SIFT は画像の特徴量である SIFT の特徴

(注1)：http://corpus-texmex.irisa.fr/
(注2)：https://nlp.stanford.edu/projects/glove/

Algorithm 6: GraphRangeSearch

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

result set $R = \emptyset$
// Global Range Search
candidate set $C \leftarrow \{s\}$
while *true* **do**
  $i \leftarrow$ the index of the first unchecked node in C
  if $i \geqq l$ **then**
    break
  $v \leftarrow i$-th candidate in C
  for $\forall u \in p.neighbors$ **do**
    $C \leftarrow C \cup \{u\}$
    if $distance(q, u) < r$ **then**
      $R \leftarrow \{u\}$
      break
  if $|R| > 0$ **then**
    break
  sort C in ascending order of the distance to q
  if $|S| > l$ **then**
    $C.resize(l)$
if $|R| = \emptyset$ **then**
  return $\emptyset$
// Local Range Search
while *true* **do**
  $i \leftarrow$ the index of the first unchecked node in R
  if checked all nodes in R **then**
    return R
  $v \leftarrow i$-th result in R
  for $\forall u \in v.neighbors$ **do**
    if $distance(q, u) < r$ **then**
      $R \leftarrow R \cup \{u\}$

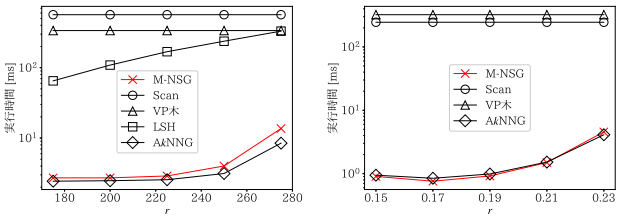
量記述子を集めたデータセットである．GloVe は Twitter で学習した自然言語の単語の埋め込み表現である．表 1 にデータセットの情報と利用する距離関数を示す．

表 1: データセット

データセット	データ数	クエリ数	d	距離関数
SIFT	1,000,000	10,000	128	ユークリッド距離
GloVe	1,183,514	10,000	25	角距離

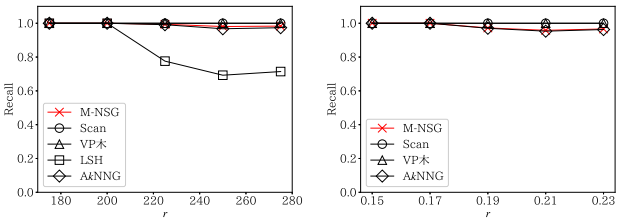
パラメータ． 以下にインデックス構築および範囲検索アルゴリズムのパラメータを示す．

- 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 kNN グラフの出次数
- 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
- ns：Navigating Node を求める際のサンプル数



(a) SIFT　(b) GloVe

図 3: r の影響（実行時間）



(a) SIFT　(b) GloVe

図 4: r の影響（Recall）

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．

表 2: パラメータ

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**
	ns	**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**
	ns	**10,000**

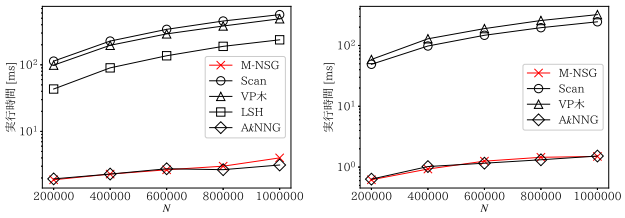
5.2 実験結果

5.2.1 r の影響

r を変化させた時の実験結果を図 3 および図 4 に示す．グラフを用いた提案アルゴリズムは他の手法と比較して高速に近似範囲検索を実行していることがわかる．すなわち，今回用いたデータセットのような高次元データにおいて，木構造やハッシュによるインデックスに比べて，距離計算を行う点を効率的に削減している．さらに提案アルゴリズムの Recall は LSH よりも安定して高く，高精度な検索が実行できている．提案アルゴリズムは，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．

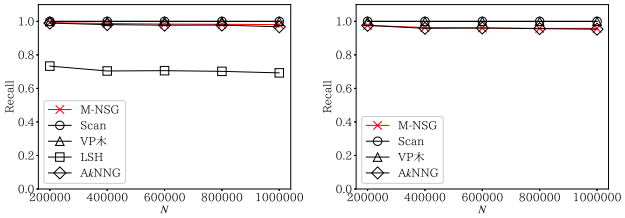
5.2.2 N の影響

N を変化させた時の実験結果を図 5 および図 6 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他の手法と比較して高速に近似範囲検索を実行している．



(a) SIFT (b) GloVe

図 5: N の影響（実行時間）

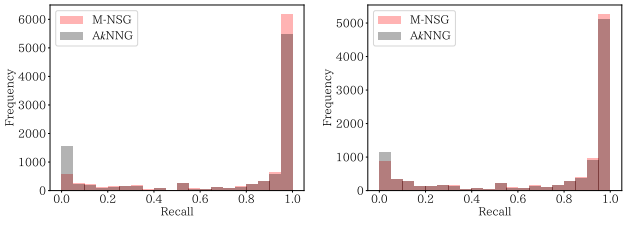


(a) SIFT (b) GloVe

図 6: N の影響（Recall）

表 3: メモリ消費量

インデックス	SIFT [MB]	GloVe [MB]
AkNNG	392	313
M-NSG	392	313



(a) SIFT (b) GloVe

図 7: Recall の分布

また，データ量が変化しても，Recall は LSH と比較して安定して高いことがわかる．

5. 2. 3 メトリック空間における Navigating Node および MRNG の有効性

図 3 および図 5 より，AkNNG と M-NSG の実行時間は差がないことがわかる．

AkNNG と M-NSG のメモリ消費量を表 3 に示す．表 3 より，メモリ消費量も差がないことがわかる．

Recall の分布を図 7 に示す．図 7 より，M-NSG を用いて検索を行った時，AkNNG を用いた時に比べて，Recall の低い検索結果の数が減り，Recall の高い検索結果が増えていることがわかる．したがって，メトリック空間における Navigating Node および MRNG により，AkNNG では到達できなかった点に到達できている．

6. 結　　論

近年，画像，音声，および埋め込み表現ベクトルなどの高次元データを対象に，高速かつ高精度な近似範囲検索が必要とされるアプリケーションが増えている．本稿では，ユークリッド空間において高速に 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．実データを用いた実験の結果から，提案アルゴリズムの有効性を確認した．

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) および JST さきがけ（JPMJPR1931）の支援を受けたものである．ここに記して謝意を表す．

文　　献

[1] Andoni, A., Indyk, P.: Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. FOCS (2006)
[2] Bentley, J.L.: Multidimensional binary search trees used for associative searching. Communications of the ACM pp. 509 – 517 (1975)
[3] Comer, D.: The ubiquitous b-tree. CSUR (1979)
[4] Datar, M., Immorlica, N., Indyk, P., Mirrokni, V.S.: Locality-sensitive hashing scheme based on p-stable distributions. SCG (2004)
[5] Dong, W., Charikar, M., Li, K.: Efficient k-nearest neighbor graph construction for generic similarity measures. WWW pp. 577 – 586 (2011)
[6] Ester, M., Kriegel, H.P., Sander, J., Xu, X.: A density-based algorithm for discovering clusters in large spatial databases with noise. KDD (1996)
[7] Fu, C., Cai, D.: Efanna : An extremely fast approximate nearest neighbor search algorithm based on knn graph. arXiv:1609.07228v3 (2016)
[8] Fu, C., Xiang, C., Wang, C., Cai, D.: Fast approximate nearest neighbor search with the navigating spreading-out graphs. PVLDB 12(5), 461 – 474 (2019)
[9] Ge, T., He, K., Ke, Q., Sun, J.: Optimized product quantization for approximate nearest neighbor search. CVPR (2013)
[10] Gionis, A., Indyk, P., Motwani, R.: Similarity search in high dimensions via hashing. PVLDB pp. 518 – 529 (1999)
[11] Hajebi, K., Abbasi-Yadkori, Y., Shahbazi, H., Zhang, H.: Fast approximate nearest-neighbor search with k-nearest neighbor graph. IJCAI (2011)
[12] Harwood, B., Drummond, T.: Fanng: Fast approximate nearest neighbour graphs. CVPR (2016)
[13] Huang, Q., Feng, J., Zhang, Y., Fang, Q., Ng, W.: Query-aware locality-sensitive hashing for approximate nearest neighbor search. PVLDB (2015)
[14] Jeff, J., Matthijs, D., Hervé, J.: Billion-scale similarity search with gpus. arXiv:1702.08734 (2017)
[15] Jegou, H., Douze, M., Schmid, C.: Product quantization for nearest neighbor search. TPAMI pp. 117 – 128 (2011)
[16] Li, H., Chan, T.N., Yiu, M.L., Mamoulis, N.: Fexipro: Fast and exact inner product retrieval in recommender systems. SIGMOD (2017)
[17] Liu, Y., Cui, J., Huang, Z., Li, H., Shen, H.T.: Sk-lsh : An efficient index structure for approximate nearest neighbor search. PVLDB (2014)
[18] Lv, Q., Josephson, W., Wang, Z., Charikar, M., Li, K.: Multi-probe lsh: Efficient indexing for high-dimensional similarity search. VLDB pp. 950 – 961 (2007)
[19] Malkov, Y., Ponomarenko, A., Logvinov, A., Krylov, V.: Approximate nearest neighbor algorithm based on navigable small world graphs. Information Systems (2014)
[20] Malkov, Y.A., Yashunin, D.A.: Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. TPAMI (2018)
[21] Qian, G., Sural, S., Pramanik, S.: A comparative analysis of two distance measures in color image databases. ICIP

(2002)
[22] Rodriguez, A., Laio, A.: Clustering by fast search and find of density peaks. Science (2014)
[23] 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 (2014)
[24] Takuma, D., Yanagisawa, H.: Faster upper bounding of intersection sizes. SIGIR (2013)
[25] Takuma, D., Yanagisawa, H.: Faster upper bounding of intersection sizes. SIGIR (2013)
[26] Yianilos, P.N.: Data structures and algorithms for nearest neighbor search in general metric spaces. SODA pp. 311 – 321 (1993)
[27] Zheng, Y., Guo, Q., Tung, A.K.H., Wu, S.: Lazylsh: Approximate nearest neighbor search for multiple distance functions with a single index. SIGMOD (2016)