メトリック空間における近接グラフを用いた高速アウトライアー検出

天方　大地†,††　　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
E-mail: †{amagata.daichi,hara}@ist.osaka-u.ac.jp

あらまし　距離に基づくアウトライアー検出は，教師なし，メトリック空間で定義可能，およびデータ分布に対する事前知識が不要という特長から多くのアプリケーションに利用されている．この技術は，距離および数に対する閾値をそれぞれ指定する必要があり，ユーザは事前に適切な閾値を把握していることは稀であるため，これらを変えながらアウトライアーを検出する必要がある．つまり，インタラクティブにアウトライアーを検出するための効率的なアルゴリズムが必要となる．本稿では，距離に基づくアウトライアー検出に対する新たなアプローチとして，近接グラフを用いたフレームワークを提案する．本フレームワークを用いることにより既存技術に対して大幅な高速化が実現できる．しかし，既存の近接グラフは本問題を想定していないため，フレームワークを最大限に効率化できない．そのため，本問題に適した近接グラフを提案する．実データを用いた実験により，提案フレームワークおよび提案グラフの有効性を示す．

キーワード　アウトライアー検出，メトリック空間

1 はじめに

アウトライアー検出は，不正行為の検出，異常検知，およびノイズ除去などの多くのアプリケーションに対する基本的なタスクである．また，距離に基づくアウトライアー検出（DOD: Distance-based Outlier Detection）[12] は，上記の要件を満たすために幅広く利用されている [5]．本稿では，DOD 問題に取り組む．

1.1 動　　機

DOD は距離に対する閾値 r および数に対する閾値 k を指定する必要がある．オブジェクト集合 P が与えられた際，$p \in P$ は，$dist(p,p') \leqq r$ を満たす p' の数が k 未満である場合にアウトライアーと定義される．ここで，$dist(p,p')$ は，p と p' 間の距離を評価している．多くのデータや距離関数で利用可能となるために，本稿ではメトリック空間で利用可能な DOD アルゴリズムを考える．これは，近年多くのアプリケーションで利用されている機械学習にも動機づけられている．

例 1. ニューラルネットワークは分類や予測タスクで多く利用されており，その精度も高い [16], [19]．高性能な分類器を訓練するためには，ニューラルネットワークは多くのデータが必要であり，また，ノイズ（アウトライアー）を取り除く必要がある．これは，ニューラルネットワークはノイズにオーバーフィットしやすいためである．機械学習アプリケーションでは，多様なデータ（時系列データ，文字列データ，および単語埋め込みベクトル等）および多様な距離関数（L_2 ノルム，編集距離，および角距離等）を処理する必要がある．そのため，これらのメトリック空間において利用可能かつ効率的なアウトライアー検出が要求される．

上記の通り，DOD アプリケーションは多くのデータを処理する必要がある．また，適切なパラメータ（r および k）は事前に知ることが困難なため，インタラクティブにパラメータをテストする必要があり，高速処理が求められる．

大容量メモリを搭載したシステムの普及により，大量のデータの場合においてもメインメモリで DOD の実行が可能となってきた．事前処理としてメインメモリ上にインデックスを構築し，このインデックスを利用することで DOD を高速に達成し得る．メトリック空間における DOD アルゴリズムの既存研究も存在する [6], [12], [17] が，これらはメインメモリ処理を考慮していないため非効率である．さらに，既存アルゴリズムは次元の呪いの影響を受けやすく，高次元データに対して性能が大幅に低下する．本稿では，任意のメトリック空間において高速なアルゴリズムを設計する．

1.2 課　　題

メトリック空間において高速なアルゴリズムを設計するためには，解決すべき課題がある．具体的には，(i) 任意の r および k に対応できる効率的なインデックスの実現，(ii) インデックスの空間効率性，および (iii) メトリック空間に対する頑強性である．

任意の r および k に対応できる効率的なインデックス． これらのパラメータは事前に知り得ないため，任意の値に対応しなければならない．既存アルゴリズムは r および k が指定された後にオンラインで簡易的なインデックスを構築している．オンラインでの有用なインデックス構築は非常に時間がかかるため実践的でなく，簡易的なインデックスでは十分に効率化できない．

空間効率性． オブジェクト p の k 最近傍を p' とすると，$dist(p,p') \leqq r$ であれば p はアウトライアーでない．そこ

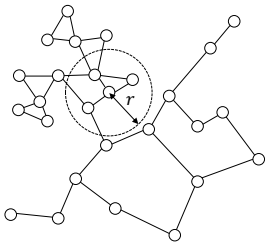


図 1　近接グラフの一例

で，各オブジェクト $p \in P$ に対して，別の全てのオブジェクトとの距離をソート済み配列で管理すれば，p がアウトライアーかどうかは $O(1)$ 時間で計算できる．しかし，この方法の空間効率は，$n = |P|$ とすると $O(n^2)$ であり，実践的でない．

メトリック空間に対する頑強性．本稿ではメトリック空間を考えており，また，機械学習等のアプリケーションでは高次元データを扱うことが一般的であることから，(メトリックを満たす) 任意のデータ空間および次元数に対して効率性が求められる．ここで，レンジクエリもあるオブジェクトがアウトライアーであるか評価できる．単純な方法は，木構造インデックスを構築し，そのインデックス上で各オブジェクトに対してレンジクエリを処理するものである．しかし，この方法は低次元データに対してのみ効率的である．

1.3 貢　　献

- **近接グラフを用いたフレームワーク**（4 章）．本稿では上記の課題を解決するソリューションとして，近接グラフを用いたフレームワークを提案する．本フレームワークはフィルタリングフェーズと検証フェーズから成る．フィルタリングフェーズにおいて近接グラフを用いてアウトライアーでないオブジェクトをフィルタリングする．近接グラフは一般的に，各オブジェクトをグラフの頂点と考え，各オブジェクトは自身との距離が短いオブジェクトへのリンク（辺）を持つ．この性質から，本フレームワークは次元の呪いの影響を受けにくく，また，アウトライアーでないオブジェクトを効率的にフィルタリングできる．下の例はその一例を示す．

例 2. 図 1 において，半径 r の円の中心の点を p とする．今 $k = 3$ とすると，p から近接グラフ（p からのリンク）を辿ることにより，p がアウトライアーでないことがわかる．

- **MRPG**（5 章）．上のフレームワークを活用するためには，フィルタリングフェーズにおける精度（フォルスポジティブ）を高く（少なく）する必要がある．これは，近接グラフにおいて，距離が r 以内となるオブジェクトへの到達可能性を高めることに相当する．これを実現するため，新たな近接グラフである MRPG（Metric Randomized Proximity Graph）を提案する．MRPG は近似 K 最近傍グラフに基づくものであり，$O(nK^2 \log K^3)$ 時間で構築できる．

- **実データを用いた評価**（6 章）．4 つの実データおよび 4 つの距離指標を用いた実験から，提案フレームワークおよび MRPG の性能を示す．提案フレームワークを用いたアプローチは既存アルゴリズムよりも高速であり，MRPG は既存の近接グラフによりも効果的にフレームワークを利用できることを確認する．

上記の内容に加えて，2 章で問題定義を行い，3 章で関連研究を紹介する．最後に 7 章にて本稿をまとめる．

2 予 備 知 識

P を n 個のオブジェクトの集合とする．あるオブジェクト $p \in P$ の隣接オブジェクトを定義する．

定義 1（隣接オブジェクト）．閾値 r が与えられた際，オブジェクト $p \in P$ と別のオブジェクト $p' \in P$ について，$dist(p, p') \leqq r$ を満たす場合，p' は p の隣接オブジェクトである．

ここで，$dist(\cdot, \cdot)$ はメトリックを満たす距離関数とする．次に距離に基づくアウトライアーを定義する．

定義 2（距離に基づくアウトライアー）．距離に対する閾値 r，数に対する閾値 k，および P が与えられた際，あるオブジェクト $p \in P$ の隣接オブジェクトの数が k 未満である場合，p は距離に基づくアウトライアーである．

本稿の問題は以下のように定義される．

問題定義．距離に対する閾値 r，数に対する閾値 k，および P が与えられた際，距離に基づくアウトライアーを全て発見する．

本稿では上の問題に対して正確かつ高速なアルゴリズムを提案する．以降では，距離に基づくアウトライアーを単にアウトライアーと呼ぶ．また，P はメインメモリで管理され，更新はないものとする．(更新がある場合，既存アルゴリズム [13] を用いて解をモニタリングすれば良い．)

3 関 連 研 究

本章では，距離に基づくアウトライアー検出に関する既存アルゴリズムおよび既存の近接グラフについて紹介する．

距離に基づくアウトライアー検出．本問題に対する最も単純なアルゴリズムは，ネステッドループを用いたものである [12]．つまり，全てのオブジェクト $p \in P$ に対して，P をスキャンし，隣接オブジェクトの数が k に達したときにスキャンを中断する．このアルゴリズムは $O(n^2)$ 時間かかり，n が大きい場合に対応できない．

SNIF [17] は，クラスタを用いることにより距離計算の回数を削減している．このクラスタは，半径 $r/2$ であり，中心のオブジェクトをランダムに選択する（クラスタの数はハイパーパラメータである）．あるオブジェクト p とあるクラスタの中心オブジェクトとの距離が $r/2$ である場合，p はそのクラスタに属する．クラスタ内のオブジェクトの数が k を超える場合，三角不等式からこのクラスタに属するオブジェクトはアウトライアーでない．もしクラスタ内のオブジェクトの数が $k + 1$ 未満である場合も，クラスタの中心オブジェクトとの距離と三角不等式を用いることにより，距離計算が不要なクラスタを枝刈りできる．DOLPHIN [6] もスキャンに基づくアルゴリズムと

なっている．このアルゴリズムでは，アクセス済みのオブジェクトをインデックスし，未アクセスのオブジェクトがアウトライアーであるかどうかの評価に用いる．現在アクセスしているオブジェクトを p とすると，このインデックスでは p からある距離以内にどれだけのオブジェクトが存在しているか把握できる．この数が k 以上であれば，p の評価を中断できる．SNIF と DOLPHIN はメトリック空間に対応できるが，I/O 数の削減を目的としており，メインメモリ処理に適していない．また，高次元空間ではクラスタや DOLPHIN の閾値の設定がルーズになり，性能が大幅に低下する．

上記のアルゴリズムに加えて，範囲検索も本問題に対応できる．本稿では，ベースラインアルゴリズムとして VP 木 [20] を用いる．VP 木は，メトリック空間における範囲検索において最も高性能なインデックスであることが示されている [9]．

近接グラフ．近年，近接グラフが近似最近傍検索において最も高性能であることが示されている [14]．これは，各オブジェクトが自身との距離が短いオブジェクトへのリンクを持つデータ構造であることに由来する．この性質は本問題においても効果的である．

最も有名な近接グラフは KGraph であり，このグラフでは，各オブジェクトが自身の近似 K 最近傍オブジェクトへのリンクを持っている．KGraph は NNDescent [11] により構築される．本稿では，このアルゴリズムを拡張し，高速に KGraph を構築する．ここで，KGraph は本問題に最適化されていないため，単純に KGraph を用いた場合は到達可能性に問題がある．例えば，$k > K$ である場合，隣接オブジェクトへの到達可能性が低い場合がある．別のモデルの近接グラフとして，NSW [8] が存在する．これは，スモールワールド性質を持たせた近接グラフである．NSW は近似最近傍検索問題のために設計されており，ランダムなオブジェクト（頂点）からのグラフ探索を想定している．本問題では，オブジェクト p の隣接オブジェクトを探索する際，p から探索すれば良いが，NSW も隣接オブジェクトへの接続性を考慮していない．

4 フレームワーク

メインアイデア．P におけるアウトライアーの割合は基本的に小さい（1%前後）[21]．つまり，大抵のオブジェクトはアウトライアーでないため，これらのオブジェクトを高速にアウトライアーではないと判定する必要がある．オブジェクト p がアウトライアーでないかどうかは，クエリを p，範囲を r とする範囲カウント問題 [1] と考えられる．この問題を効率的に解くことにより，アウトライアーでないオブジェクトを高速にフィルタできる．本稿では，各オブジェクトが距離が短いオブジェクトへのリンクを保持している近接グラフに着目する．この特徴を利用すれば，近接グラフにおいて p からグラフを辿ることにより，p の類似オブジェクトを効率的に発見できることが分かり，さらに，次元の呪いも受けにくい．

1：p から r 以内に存在するオブジェクトの数をカウントする問題．本稿ではカウントが k になれば計算を終了する．

Algorithm 1: Greedy-Counting

Input: p_i, r, k, and a proximity graph G

```
1  count ← 0
2  Q ← {v_i}
3  Check v_i as visited
4  while Q ≠ ∅ do
5      v ← the front of Q
6      Q ← Q\{v}
7      for each v' ∈ v.E do
8          if v' has not been checked as visited then
9              Check v' as visited
10             if dist(p, p') ≦ r then
11                 count ← count +1
12                 if count = k then
13                     break
14                 Q ← Q ∪ {v'}
15             else
16                 if p' is a pivot then
17                     Q ← Q ∪ {v'}
18     if count = k then
19         break
20 return count
```

このアイデアに基づき，本稿では近接グラフを用いたフレームワークを提案する．本フレームワークはフィルタリングフェーズと検証フェーズから成る．

フィルタリングフェーズ．このフェーズでは，オフライン処理で構築されている近接グラフ G を用いてアウトライアーでないオブジェクトをフィルタする．これを効率的に行うため，Greedy-Counting（アルゴリズム 1）を提案する．G における頂点 v をオブジェクト p に相当すると考え，$v.E$ を v が持つつリンクの集合とする．Greedy-Counting は，v から G を辿る貪欲法である．具体的に，まず $v.E$ に $dist(p, p') \leqq r$ となるオブジェクト p' があり，かつ p'（v'）が未訪問であれば，カウントを 1 増やし，v' をキュー Q に入れる．次に，Q の先頭をポップし，v と同様の処理を行う．（例外が 16 行にあるが，これは後に説明する．）Greedy-Counting は，カウントが k になるか，Q が空になれば終了する．また，このアプローチは，以下の性質を持つ．

補助定理 1．フィルタリングフェーズにおいてフォルスネガティブが生じることはない．

証明．ページ数の都合上，全ての証明を割愛する． □

検証フェーズ．フィルタリングフェーズにおいてフィルタされなかったオブジェクトの集合を P' とする．これらのオブジェクトは，以下の方法により隣接オブジェクトの数をカウントする．

- 低次元データの場合，VP 木を構築し，各オブジェクトに対して VP 木上で範囲カウンティングを行う．
- それ以外の場合，シーケンシャルスキャンにより範囲カ

ウンティングを行う．
いずれの場合においてもカウントが k に達した場合，範囲カウンティングを中断する．

分析．これ以降，データの次元数を定数とする．各オブジェクトに対して，Greedy-Counting が終了するまでにアクセスしたオブジェクトの数の平均を ρ とする．このとき，フィルタリングフェーズは $O(\rho n)$ 時間かかる．また，フォルスポジティブとアウトライアーの数をそれぞれ f および t とすると，検証フェーズは $O((f+t)n)$ 時間かかる．アウトライアー検出に有効な k（大きすぎない k）が指定されたとき，ρ は小さく，$\rho = O(k)$ であるため，提案フレームワークの時間計算量は $O((f+t)n)$ となる．

マルチスレッディング．提案フレームワークでは各オブジェクトを独立に評価するため，マルチスレッディングによる高速化が可能である．つまり，フィルタリングフェーズおよび検証フェーズにおいて，各スレッドは割り当てられたオブジェクトを評価すれば良い．ここで，アウトライアーの評価はアウトライアーでないオブジェクトよりも時間を要するが，どのオブジェクトがアウトライアーか事前に知り得ないため，負荷分散が理論的に不可能である．そのため，マルチスレッディングの際は P をランダムに分割する方法を採用する．

5 MRPG

提案フレームワークは任意の近接グラフで利用可能であるが，計算時間を削減するためにはフィルタリングフェーズにおけるフォルスポジティブを削減する必要がある．これにより，検証フェーズにかかる時間が短縮されるためである．フォルスポジティブを削減するためには，任意のオブジェクト p に対して Greedy-Counting が p の隣接オブジェクトを辿れる近接グラフが必要になる．これは，任意のオブジェクト間に単調経路（2 つのオブジェクト間の経路において，ある始点と中継オブジェクトとの距離が単調に大きくなる経路）[10] があれば可能である．しかし，このようなグラフを作成するには $\Omega(n^2)$ 時間必要であり，実践的でない．

そこで，本稿では上のグラフを近似した近接グラフである MRPG（Metric Randomized Proximity Graph）を提案する．MRPG は 2 つのアイデアから構築される．1 つ目は，近似 KNN グラフ（AKNNG）の利用である．AKNNG では，各オブジェクトは自身に（最も）類似したオブジェクトとのリンクを持つため，隣接オブジェクトへのアクセス効率が良い．もう一方は，各サブ空間からランダムにピボットを選び，ピボットを利用して単調経路を作成するものである．これにより，隣接オブジェクトへの到達可能性が向上する．MRPG は以下のステップにより構築される．

（1） NNDescent+：まず，AKNNG を構築する．これを効率的に行うため，既存 AKNNG 構築アルゴリズムである NNDescent を拡張する．

（2） Connect-SubGraphs：AKNNG は連結グラフでない場合がある．その場合，MRPG を連絡グラフにする．

（3） Remove-Detours：次に，遠回りとなる経路を修正し，単調経路を作成する．任意のオブジェクト間に単調経路を作成するコストが大きすぎるため，近似ヒューリスティックを提案する．

（4） Remove-Links：最後に，オブジェクトへの無駄なアクセスを削減するために不要なリンクを削除する．

5.1 NNDescent+

MRPG は AKNNG から構築されるため，AKNNG の効率的な構築アルゴリズムが必要となる（KNNG の構築時間は $O(n^2)$ のため考えない）．AKNNG の構築アルゴリズムの代表例は NNDescent [11] である．まず，このアルゴリズムを紹介する．

NNDescent．このアルゴリズムの基となるアイデアは，オブジェクト p とそれに類似するオブジェクト p' が与えられた際，p' に類似するオブジェクトは p にも類似する傾向にあるというものである．つまり，p の AKNN は，p および p' に類似するオブジェクトを探索することにより得られる．K および P が与えられた際，NNDescent[2]の具体的な処理は以下の通りである．

（1） 各オブジェクト $p \in P$ に対して，ランダムな K 個のオブジェクトを初期の AKNN とする．

（2） 各オブジェクト $p \in P$ に対して，類似オブジェクトリストを計算する．このリストには，p の現在の AKNN および逆 AKNN が含まれる．今，p' を p の類似オブジェクトリストに含まれるオブジェクトとすると，全ての p' の類似オブジェクトリストにアクセスし，AKNN を更新するオブジェクトが存在すれば，AKNN を更新する．

（3） 上の処理を繰り返し行い，更新が起こらなくなった時点で終了する．

上のアルゴリズムの時間計算量は以下の通りである．

定理 1．NNDescent の時間計算量は $O(nK^2 \log K)$ となる．

NNDescent+．NNDescent が高精度の AKNNG を構築することは実験的に示されているものの，計算処理に関して以下の問題点が挙げれれる．

- 初期に完全にランダムなオブジェクトを AKNN にするため，2 番目の処理の回数が増加してしまう．
- 更新が起こらないオブジェクトの類似オブジェクトリストにアクセスしてしまう．

上記の問題は無駄な距離計算を起こすため，本稿では NNDescent を拡張し，これらの問題を解決する．

VP 木に基づくデータ集合分割．無駄な距離計算を削減するためには，各オブジェクトに類似するオブジェクトを早期に発見する必要がある．VP 木の構築方法を応用することにより，これを実現する．

VP 木は P を再帰的に分割することにより構築される．今 VP 木のあるノードが P を持つとする．ノードがもつオブジェクトの数がキャパシティ c を超える場合，このノードは左およ

2：本稿では，並列処理の有用性から，文献 [11] における基本版のアルゴリズムを採用する．

び右ノードの 2 つに分割される．具体的には，p を P の中のランダムなオブジェクトとすると，p とそれ以外のオブジェクトとの距離を計算し，中央値を得る．あるオブジェクト $p' \in P$ との距離が中央値以下であれば左ノードに，そうでなければ右ノードに割り当てられる．この操作を，分割が起こらなくなるまで繰り返す．

今，$c = O(K)$ とし，親ノードの左ノードとなる葉ノードについて考える．このノードが保持するオブジェクトの集合を P' としたとき，超球に基づく分割方法から P' に含まれるオブジェクトは互いに類似する傾向にある．そのため，P' に含まれるオブジェクトは，P' における K-NN を初期の AKNN とする．このアプローチにより，ランダムに比べてより精度の高い初期 AKNN が得られる．また，NNDescent の効率性も失わない．

補助定理 2. NNDescent+による初期化は $O(n \log n + 2^{\log n} K^2 \log K)$ 時間かかる．

$O(2^{\log n}) \ll O(n)$ であるため，このアプローチは NNDescent の 2 番目の処理よりも効率的である．ここで，分割アルゴリズムのランダム性から，P' に含まれないオブジェクトが存在する可能性がある．そのため，この分割は定数回行う．この処理を NNDescent の最初の処理と置換する．

また，左ノードを葉ノードとするノードをピボットとし，これ以降の処理で用いる．超球に基づく分割のため，ピボットは様々な部分空間に分散することになる．

AKNN の更新がないオブジェクトのスキップ. 類似オブジェクトリストを作成する際，NNDescent+では AKNN に更新があったオブジェクトのみリストに追加する.（各オブジェクトの AKNN の更新状況を管理するため，ハッシュテーブルを用いる.）これにより，無駄な距離計算が削減されるため，2 番目の処理が効率化される．

正確な K-NN の検索. 上の 2 つのアプローチにより，2 番目の処理回数および距離計算回数が削減されるが，K-NN への距離が比較的大きい疎なオブジェクトは精度が低くなる傾向がある．分割アルゴリズムは距離が小さいオブジェクトをクラスタリングしていることに相当しているため，疎なオブジェクト集合をクラスタリングできない．これにより偏りのあるクラスタができてしまう可能性があり，2 番目の処理において類似オブジェクトリストに偏りが生じてしまう．この場合，疎なオブジェクトは精度の高い K-NN を得ることが難しい．この問題を解決するため，疎なオブジェクトに対しては正確な K-NN を計算する．

NNDescent における 2 番目の処理が終わった後，P を AKNN までの距離の合計が大きい順にソートする．この値（距離の合計）が大きいオブジェクトは疎なオブジェクトである可能性が高く，AKNN の精度が高くないと考える．そこで，P の先頭の m 個のオブジェクトに対して正確な K-NN を計算する．また，m は $m \ll n$ となる定数とする．このとき，このアプローチの計算量は $O(n \log n)$ である．

以上の拡張により，NNDescent+は高速に AKNNG を構築する．また，以下が成り立つ．

補助定理 3. NNDescent+の時間計算量は $O(nK^2 \log K)$ である．

NNDescent+の理論的な速度は NNDescent と変わらないが，多くの場合で NNDescent+は NNDescent よりも実践的に高速である．

5.2 サブグラフ間の連結

次に AKNNG を連結グラフにする．これは，$K \ll n$ であるため，AKNNG が非連結なサブグラフから構成される場合があるからである．この場合，Greedy-Counting が隣接オブジェクトに到達できない可能性が高まってしまう．この問題を解決するアルゴリズムとして，Connect-SubGraphs を提案する．Connect-SubGraphs は 2 フェーズから成る．

逆 AKNNG フェーズ. まず，逆 AKNN を利用する．オブジェクト p が別のオブジェクト p' の AKNN に含まれている場合，p は p' へのリンクを追加する．AKNNG は有向グラフと考えられるが，この操作は AKNNG を無向グラフにすることに相当する．このアプローチはシンプルであるものの，逆 AKNN も距離が短い場合が多いため，隣接オブジェクトへの到達可能性を向上できる．

BFS フェーズ. 次に，幅優先探索（BFS）を用いて AKNNG が非連結グラフであるか確認する．ランダムなオブジェクトを始点として BFS を開始したとき，アクセスできなかったオブジェクトが存在すれば，AKNNG は非連結グラフである．

P' をアクセスできなかったオブジェクトの集合とする．このフェーズでは，P' 内のあるピボットと $P \backslash P'$ 内のあるピボット間に経路を作成する．P' 内のランダムなピボットを v'_{piv}，$P \backslash P'$ 内のランダムなピボットの集合を V_{piv} とする（$|V_{piv}|$ を小さな定数とする）．そして，v'_{piv} に対して $P \backslash P'$ における近似最近傍オブジェクトを検索し，それらの間にリンクを作成する．これにより，可能な限り距離が短いオブジェクト間にリンクを作成できる．

近似最近傍オブジェクトの検索は，文献 [15] で提案されている貪欲法を用いる．この貪欲法の入力は，クエリオブジェクト（v'_{piv}），開始点（$v \in V_{piv}$），および近接グラフである．この貪欲法は，v から検索を開始し，現在の頂点が持つリンクの中で v'_{piv} との距離が最も短いオブジェクトを検索し，そのオブジェクトに移動することを繰り返すことで近似最近傍オブジェクト v_{ann} を得る．この操作を V_{piv} 内の全てのピボットに対して実行し，v'_{piv} との距離が最も短い近似最近傍オブジェクト v_{res} を求める．その後，v'_{piv} と v_{res} との間にリンクを作成し，P' 内のランダムなオブジェクトから BFS を再開する．これを BFS が全てのオブジェクトにアクセスするまで繰り返す．

Connect-SubGraphs では，近似最近傍検索にホップ数の制限を設ける．これは，v'_{piv} との距離が大きいピボットを無視するためである．また，この操作の時間計算量を $O(K)$ にでき，以下が成り立つ．

補助定理 4. Connect-SubGraphs の時間計算量は $O(nK)$ である．

5.3 遠回り経路の削除

オブジェクト p とその隣接オブジェクト p' への経路が単調経路でない（遠回りである）場合を考える．このとき，Greedy-Counting は p から p' に到達できない場合がある．例えば，p_1 と p_2 の 2 つのオブジェクトが与えられた際，$dist(p_1, p_2) \leqq r$ とする．ここで，p_1 から p_2 への経路が 1 つしかなく，$p_1 \to p_3 \to p_2$ とする．もし $dist(p_1, p_3) > r$ であると，Greedy-Counting は p_1 から p_2 に到達できない．これはフィルタリングフェーズにおけるフォルスポジティブを増やしてしまうため，単調経路を作成することが望ましい．任意の 2 頂点間に単調経路が存在するグラフは，Monotonic Search Graph（MSG）と呼ばれる．

MSG の構築．理論的に MSG の構築には $\Omega(n^2)$ 時間かかる．これは，全てのオブジェクトが他の全てのオブジェクトとの経路を確認する必要があるためである．以降では，実践的な MSG の構築について考え，本稿では BFS に基づいてあるオブジェクトと他のオブジェクト間に単調経路があるか確認する Get-Non-Monotonic() を提案する．

Get-Non-Monotonic()．オブジェクト p_1 が与えられたとする．この関数は，p_1 から BFS を実行する．今，オブジェクト p_3 にアクセスし，その経路が $p_1 \to p_2 \to p_3$ であったとする．もし $dist(p_1, p_2) > dist(p_1, p_3)$ であれば，この経路は遠回り経路である．この関数では，p_1 からの単調経路が確認できなかったオブジェクトを p_1 との距離とともに配列 A_1 で管理する．全てのオブジェクトにアクセスした後，A_1 を距離の小さい順にソートする．

上の処理を全てのオブジェクトに対して実行すると，各オブジェクトに対して A_i が得られる．各 $j \in [1, s-1]$ に対して，$A_i[j]$ と $A_i[j+1]$ との間にリンクを作成する（s は A_i のサイズである）．これにより MSG を構築できるが，その計算コストは大きい．

定理 2．MSG の構築には，$O(n^2(\log n + K))$ 時間かかる．

ヒューリスティックによる近似．定理 2 から，MSG の構築は実践的でないことが分かる．ここで，r と k は一般的に小さい [13], [18], [21] ため，実践的には距離が小さいオブジェクトへの単調経路があれば良い．そこで，(i) AKNNG では，p との距離が小さいオブジェクトは p から数ホップに存在する，(ii) p と p に類似するオブジェクト p' が与えられた際，p' に類似するオブジェクトは p に類似しやすい（NNDescetn のアイデア），という性質を利用する．具体的には，(i) および (ii) に現れるオブジェクトにアクセスできれば p に対して必要なリンクを作成できるというアイデアに基づいたヒューリスティック Remove-Detours を設計する．

まず，(単調経路を作る）ターゲットオブジェクトとして $|P'|$ 個のオブジェクトをランダムにサンプルする．次に，各 $p \in P'$ に対して以下を行う．

- p から 3-hop BFS（p から 3 ホップで中断する BFS）を実行し，p からの単調経路がないオブジェクトを管理する．これは，ホップ数制限のある Get-Non-Monotonic() である．
- p との距離が小さいピボット（p から 1 ホップ以内に存在するものを除く）をランダムに $|P_{piv}|$ 個サンプルし，各 $p' \in P_{piv}$ に対して p' から 2-hop BFS を実行する．つまり，p' から BFS を開始し，p' から 2 ホップ以内に存在するオブジェクトと p との距離を計算する．

その後，MSG の構築と同様の方法でリンクを追加する．

Remove-Detours では，$|P'| = O(\frac{n}{K})$ および $|P_{piv}| = O(K)$ とする．また，Get-Non-Monotonic() で管理する A のサイズを $|A| \leqq O(K^2)$ とする（p との距離の小さいものだけ管理する）．これにより，次の補助定理が得られる．

補助定理 5．Remove-Detours の時間計算量は $O(nK^2 \log K^3)$ である．

5.4 不要なリンクの削除

各オブジェクトは自身に類似するオブジェクトへのリンクを持っているため，p_1 にリンクがあるオブジェクト p_2 は，p_1 と共通のリンクを持つ場合がある．例えば，p_1 と p_2 が p_3 へのリンクを持っているとする．このとき，Greedy-Counting は，p_3 に少なくとも 2 回アクセスすることになる．このような無駄なアクセスが何度も起きると計算時間が増加してしまう．これを避けるため，ピボットを利用して無駄なリンクを削除する Remove-Links を提案する．

ピボットでないオブジェクト p について考える．もし p がピボット p' へのリンクを持っている場合，p は，p と p' に共通するリンクを削除する．Remove-Links は，全てのピボットでないオブジェクトに対して上の操作を実行する．

補助定理 6．Remove-Links の時間計算量は $O(nK)$ である．

この削除を行うため，アルゴリズム 1 における 16–17 行が必要となる．

5.5 理論的解析

最後に，MRPG 構築に必要な計算量と MRPG のメモリ使用量を分析する．補助定理 3–6 から，

定理 3．MRPG の構築には，$O(nK^2 \log K^3)$ 時間かかる．

また，補助定理 4 および 5 から，

定理 4．MRPG のメモリ使用量は，$O(nK)$ である．

6 評価実験

本実験は，Dual Intel Xeon E5-2687w v4 processor（3.0GHz，12 コア）および 512GB RAM を搭載した計算機により行った．全てのアルゴリズムは C++で実装され，g++ 7.4.0（-O3）によりコンパイルされている．マルチスレッディングは OpenMP を用いた．

データ．本実験は，Glove [1]，HEPMASS [2]，SIFT [3]，および Words [4] の 4 つの実データを用いた．表 1 にこれらのデータの統計と使用した距離関数を示す．

アルゴリズム．本実験は，以下のアルゴリズムを評価した．

表 1　データの統計と距離関数

データ	データ数	次元数	距離関数
Glove	1,193,514	25	角距離
HEPMASS	7,000,000	27	L_1 ノルム
SIFT	1,000,000	128	L_2 ノルム
Words	466,551	1–45	編集距離

表 2　デフォルトパラメータ

データ	r	k	アウトライアーの割合
Glove	0.25	20	0.55%
HEPMASS	15	50	0.65%
SIFT	320	40	1.04%
Words	5	15	4.16%

- 既存アルゴリズム：Nested-loop [7]，SNIF [17]，DOLPHIN [6]，および VP 木 [20].
- 近接グラフ：NSW [15]，KGraph [11]，および MRPG.

HEPMASS および Words の場合，検証フェーズで VP 木を用いた.

前処理時間の制限を 12 時間，アウトライアー検出（オンライン処理）の時間制限を 8 時間とした.

パラメータ. 表 2 に本実験におけるデフォルトパラメータを示す．前処理およびアウトライアー検出ではそれぞれ 48 および 12 スレッドを用いた．また，KGraph および MRPG において $K = 25$ とし，NSW は KGraph のメモリ使用量と同程度になるようにリンクの数を指定した.

6.1　前処理性能の評価

まず，NSW，KGraph，および MRPG の前処理時間を評価した.（VP 木はいずれのデータにおいても 60 秒以内に処理を終えており，詳細は割愛する.）表 3 に結果を示す．Words を除き，MRPG の前処理時間が最も短い．NSW はインクリメンタルに頂点を追加するため，マルチスレッドを利用できず，大量のデータにスケールしない．また，KGraph および MRPG の前処理時間はそれぞれ $O(nK^2 \log K)$ および $O(nK^2 \log K^3)$ にもかかわらず，MRPG の方が実践的には早い．これは，NNDescent+の効率性が理由である．表 4 に，Glove における前処理時間の内訳を示しているが，NNDescent+が効率的に AKNNG を構築していることが分かる．また，それ以外の関数により効率的に単調経路を作成していることも分かる.

表 3　前処理時間 [秒]

データ	NSW	KGraph	MRPG
Glove	2333.47	923.83	**755.54**
HEPMASS	44802.80	7935.25	**4345.63**
SIFT	4910.94	929.478	**723.75**
Words	871.27	**455.15**	707.08

一方，Words では KGraph の構築の方が早い．Words では編集距離を用いたが，これは次元数の自乗のコストがかかる．NNDescent+では正確な K-NN を計算するオブジェクトが存在するが，これらの次元数が多いことを確認した．そのため，

表 4　Glove における前処理時間の内訳 [秒]

アルゴリズム	KGraph	MRPG
NNDescent(+)	923.83	464.34
Connect-SubGraphs	-	20.36
Remove-Detours	-	278.21
Remove-Links	-	19.44

正確な K-NN を計算する処理に時間を要し，MRPG の構築に時間がかかる.

6.2　アウトライアー検出性能の評価

次に，アウトライアー検出に要した時間を評価する．表 5 に実行時間を示す.

既存アルゴリズムとの比較. まず，提案フレームワークを用いたアプローチ（NSW，KGraph，および MRPG）と既存アルゴリズムを比較したとき，提案フレームワークを用いたものが既存アルゴリズムよりも高速であることが分かる．これは不要な距離計算を大幅に削減しているためである．近接グラフでは，各オブジェクトが類似するオブジェクトへのリンクを持っているため，隣接オブジェクトのカウントに要する時間が既存アルゴリズムよりも短い．例えば，MRPG は既存アルゴリズムの中で最速なものに比べて，Glove，HEPMASS，SIFT，および Words においてそれぞれ 18.4 倍，13.5 倍，9.9 倍，および 2.7 倍早い.

MRPG と既存近接グラフとの比較. 次に，MRPG と NSW および KGraph を比較すると，MRPG は全てのデータにおいて最速であることが分かる．これは，隣接オブジェクトへの到達可能性を向上していることが理由である．詳細は割愛するが，フィルタリングフェーズにおけるフォルスポジティブの数は MRPG が最も少ない．そのため，検証フェーズに要する時間を削減しており，アウトライアー検出の性能が最も良い.

k の影響. 最後に，アウトライアーの割合の影響を調べるため，k を変えて実験を行った．表 5 から提案フレームワークの方が既存アルゴリズムよりも性能が良いことは明らかなので，提案フレームワークのみを評価した．図 2 にその結果を示す（r を変えた結果も同様である）.

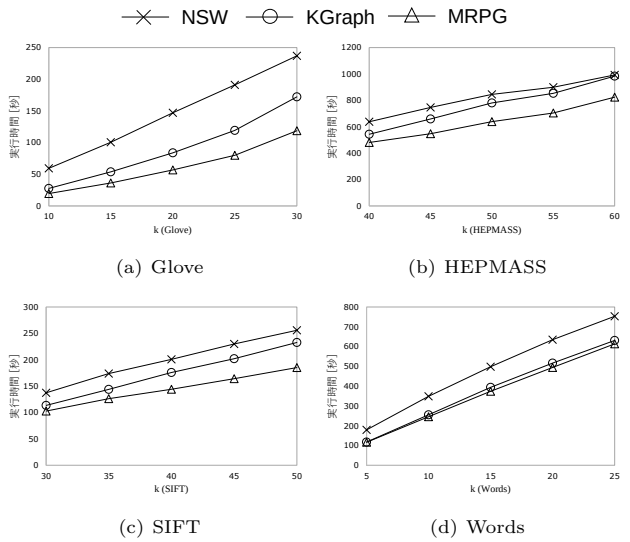
提案フレームワークは k が増加すると実行時間が増加する．これは k が増えるとグラフ探索に必要な頂点数が増加すること，および，k が増えるとアウトライアーの割合が増加するため，検証フェーズにかかる時間が増加することが理由である．また，k によらず MRPG の性能が最も良い．これは，k が増加しても隣接オブジェクトへの到達可能性を保持していることを示している.

7　おわりに

本稿では，メトリック空間における距離に基づくアウトライアー検出問題に取り組んだ．新たな解法として，近接グラフを用いたフレームワークを提案した．さらに，このフレームワークを有効活用するため，MRPG を提案した．実データを用い

表 5　実行時間 [秒]（太字は最速を示す.）

データ	Nested-loop	SNIF	DOLPHIN	VP 木	NSW	KGraph	MRPG
Glove	1045.47	1222.43	9277.89	1398.92	147.00	83.82	**56.80**
HEPMASS	17295.40	20360.80	NA	8597.23	845.411	780.194	**638.83**
SIFT	1427.74	1507.58	8684.08	2005.27	200.89	175.88	**144.106**
Words	1844.65	2086.50	7061.50	1021.39	498.34	393.95	**374.08**



(a) Glove　(b) HEPMASS

(c) SIFT　(d) Words

図 2　k の 影 響

た実験により，提案フレームワークは既存アルゴリズムよりも高速であり，MRPG は既存近接グラフよりも高性能であることを確認した.

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

文　献

[1] `https://nlp.stanford.edu/projects/glove/`.
[2] `https://archive.ics.uci.edu/ml/index.php`.
[3] `http://corpus-texmex.irisa.fr/`.
[4] `https://github.com/dwyl/english-words`.
[5] C. C. Aggarwal. Outlier analysis. In *Data mining*, pages 237–263, 2015.
[6] F. Angiulli and F. Fassetti. Dolphin: An efficient algorithm for mining distance-based outliers in very large datasets. *TKDD*, 3(1):4, 2009.
[7] S. D. Bay and M. Schwabacher. Mining distance-based outliers in near linear time with randomization and a simple pruning rule. In *KDD*, pages 29–38, 2003.
[8] M. Boguna, D. Krioukov, and K. C. Claffy. Navigability of complex networks. *Nature Physics*, 5(1):74, 2009.
[9] L. Chen, Y. Gao, B. Zheng, C. S. Jensen, H. Yang, and K. Yang. Pivot-based metric indexing. *PVLDB*, 10(10):1058–1069, 2017.
[10] D. Dearholt, N. Gonzales, and G. Kurup. Monotonic search networks for computer vision databases. In *ACSSC*, volume 2, pages 548–553, 1988.
[11] W. Dong, C. Moses, and K. Li. Efficient k-nearest neighbor graph construction for generic similarity measures. In *WWW*, pages 577–586, 2011.
[12] E. M. Knorr and R. T. Ng. Algorithms for mining distance-based outliers in large datasets. In *VLDB*, volume 98, pages 392–403, 1998.
[13] M. Kontaki, A. Gounaris, A. N. Papadopoulos, K. Tsichlas, and Y. Manolopoulos. Continuous monitoring of distance-based outliers over data streams. In *ICDE*, pages 135–146, 2011.
[14] W. Li, Y. Zhang, Y. Sun, W. Wang, M. Li, W. Zhang, and X. Lin. Approximate nearest neighbor search on high dimensional data-experiments, analyses, and improvement. *IEEE Transactions on Knowledge and Data Engineering*, 2019.
[15] Y. Malkov, A. Ponomarenko, A. Logvinov, and V. Krylov. Approximate nearest neighbor algorithm based on navigable small world graphs. *Information Systems*, 45:61–68, 2014.
[16] R. C. Marcus and O. Papaemmanouil. Plan-structured deep neural network models for query performance prediction. *PVLDB*, 12(11):1733–1746, 2019.
[17] Y. Tao, X. Xiao, and S. Zhou. Mining distance-based outliers from large databases in any metric space. In *KDD*, pages 394–403, 2006.
[18] L. Tran, L. Fan, and C. Shahabi. Distance-based outlier detection in data streams. *PVLDB*, 9(12):1089–1100, 2016.
[19] Y. Wang, W. Liu, X. Ma, J. Bailey, H. Zha, L. Song, and S.-T. Xia. Iterative learning with open-set noisy labels. In *CVPR*, pages 8688–8696, 2018.
[20] P. N. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *SODA*, volume 93, pages 311–21, 1993.
[21] S. Yoon, J.-G. Lee, and B. S. Lee. Nets: extremely fast outlier detection from a data stream via set-based processing. *PVLDB*, 12(11):1303–1315, 2019.