CD-001 マルチメトリック空間における効率的な近似最近傍探索

植村 玲央, 天方 大地, 原 隆浩
大阪大学
{uemura.reon,amagata.daichi,hara}@ist.osaka-u.ac.jp

アブストラクト

多くのオブジェクトはマルチモーダルデータ（画像，動画，およびテキスト等から成るデータ）であり，異なるメトリックを組み合わせたオブジェクトの探索は，情報検索や機械学習に多く用いられている．その中でもマルチメトリック最近傍探索は，異なるメトリックを組み合わせた距離によって類似性を測定し，クエリに最も類似したオブジェクトを発見するものである．一般的に，オブジェクトがもつ画像やテキストは高次元ベクトルで表現されており，この場合に厳密なマルチメトリック最近傍を出力するアルゴリズムとしては線形探索が最速であること知られているが，大量のデータに対する探索遅延は非常に大きい．そのため，高速に高精度な近似解を求めるアルゴリズムがこれまでに開発されているものの，既存アルゴリズムはメトリック間の重みを考慮できないものや無駄な距離計算を削減できていない．本研究では，上記の課題を解決するために，近接グラフを用いた新しいマルチメトリック最近傍探索アルゴリズムを提案する．このアルゴリズムでは，代表ノードまたは VP 木から計算した近似最近傍点からグラフ探索を行い，その解を各メトリックの近接グラフ間で共有することにより，探索範囲を削減する．実データを用いた評価実験により，提案アルゴリズムの有効性を示す．

1　はじめに

k-最近傍探索問題は，与えられたクエリとオブジェクト集合内の各オブジェクトに対して，あるメトリック（例えば L_p ノルム，角距離，および編集距離）に基づいて類似性を評価し，最も類似した k 個のオブジェクトを見つける．この問題は多くの既存研究があり，情報検索 [25]，パターンマッチング [21]，DNA 解析 [26]，地理情報システム [33]，および機械学習 [29] 等の幅広い応用で用いられている．

1.1　研究動機

近年では，オブジェクトの量のみならず，モダリティも増している．具体的には，多くのオブジェクトはマルチモーダルデータ（画像，動画，およびテキスト等から成るデータ）であり [36]，オブジェクトを構成する各要素はそれぞれ異なるメトリック空間で表現される．このようなオブジェクトは，最近注目を集めているマルチモーダル検索において非常に重要なデータである．例えば，大規模ヴィジョン言語モデル（VLM）[32] は text-to-image を高精度に処理しており，Sora[1]は，テキストから動画を生成（text-to-video）しているが，これらを支えるのはテキストと画像等のペアをもつ大量のマルチモーダルデータから成る訓練データである．また，あるマルチモーダルデータに類似するオブジェクトは，いずれかのメトリックでのみ類似しているわけではない．つまり，オブジェクト間の類似性を評価するためには単一のメトリックでは意味をなさないため，複数のメトリックに着目する必要がある．しかし，既存研究の多くはマルチモーダルデータを想定しておらず，単一のメトリックのみの最近傍探索に焦点を当てている．

本研究では，異なるメトリックを組み合わせた距離によって類似性を測定し，クエリに最も類似したオブジェクトを発見するマルチメトリック最近傍探索問題に取り組む．これは，オブジェクト集合 O，クエリオブジェクト q，k，および各メトリックに対する重み $W = \langle w_i, \ldots, w_m \rangle$ が与えられたとき，q との重み付き距離が最も小さい k 個のオブジェクト $\in O$ を出力する問題である（詳細は2章で述べる）．本問題の実応用例として，Azure AI Search[2]が挙げられる．

[1]https://openai.com/sora

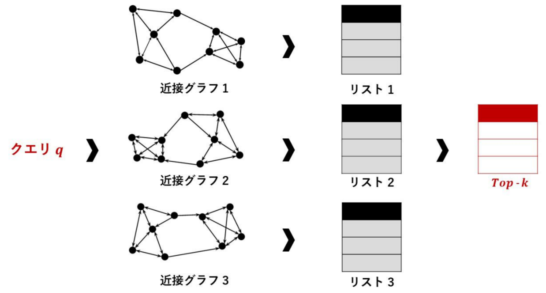

図 1: 分割型アプローチの一例．各メトリックで近接グラフを用いた近似マルチメトリック最近傍探索.

1.2　課題

マルチメトリック最近傍探索問題に対する素朴なアルゴリズムは線形探索である．近年の埋込技術の普及により，画像やテキスト等は高次元ベクトルで表現されることが一般的であり，このようなオブジェクトに対して厳密解を出力するアルゴリズムとしては線形探索が最速であることが知られている [24]．しかし，線形探索は大量のオブジェクトに対して探索を短時間で行えない．そのため，近似解を求める近似（マルチメトリック）最近傍探索の研究が多く行われている．単一のメトリックに着目した高次元空間での近似最近傍探索のアプローチとして，Locality Sensitive Hashing（LSH）[20, 27]，量子化 [4, 23]，木構造 [17]，および近接グラフ [22, 30, 35] がある．この中で，近接グラフを用いたアプローチが最も高速かつ高精度であることが実践的に示されている．

マルチメトリック最近傍探索問題では，複数のメトリックを統合して一つのインデックスを作る統合型 [8, 9, 14, 18] と，メトリック毎にインデックスを構築する分割型 [7, 18, 28, 36] の二つのアプローチがある．統合型では一つのインデックスで全てのデータを管理するため，重みを考慮できない．例えば，あるメトリックの重みを 0，つまり完全に無視する場合を考える．ここで，インデックス構築時には探索における重みはわかっていないため，重みを均等にしてインデックスを構築するしかない．そのため，探索時に重みが 0 のメトリックがあったとしても，均等な重みの前提でインデックスを構築しているため，そのメトリックを無視できない．分割型では，メトリック毎にインデックスを構築し，各メトリックのインデックスを用いた探索結果をマージする．分割型のイメージを図1に示す．分割型では複数のインデックスを構築するために，インデックス構築コストは統合型と比較して大きくなってしまうが，探索時に重みを考慮できる．

以上の観測から，それぞれのメトリックにおいて近接グラフを用いる分割型のアプローチが有効であることが分かる．このアプローチでは，データベースをメトリック毎に分割し，それぞれのメトリックで近接グラフを構築す

[2]https://azure.microsoft.com/en-us/products/ai-services/ai-search

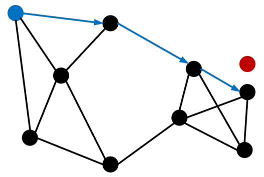

図 2: 貪欲法による探索の一例．赤点がクエリ，青点が初期探索ノード，青矢印が探索ルートを表している．

る．近接グラフにおいて，ノードはオブジェクトであり，距離が短いノード間に辺が作られる．各メトリックの近接グラフ上で，あるノードから貪欲法を用いて探索を行い，各近接グラフでの k-最近傍を計算する．そして，それらをマージし，最終的な解を計算する．貪欲法では探索ノードの隣接ノードの内，未探索のノードとクエリの距離を計算し，最もクエリに近い隣接ノードを次の探索ノードとする．図2に一例を示す．

ここで，このアプローチには 2 つの問題がある．1 つ目は，探索を開始する初期ノード（初期探索ノード）に関する問題である．最近傍探索は距離計算がボトルネックであり，距離計算回数を削減することは探索時間の削減に直結する．貪欲法による最近傍探索では，初期探索ノードがクエリから遠ざかるほど，ホップ数が増加する．探索では各ホップで隣接ノードの数だけ距離計算を行うため，ホップ数の増加に伴って距離計算回数も増加する．したがって，初期探索ノードがクエリに近いほど探索時間は削減される．2 つ目はメトリック間における解の共有に関する問題である．分割型では，各インデックス上で探索を行い，結果をマージする．この過程において，各メトリック間で独立して探索を行う場合，各メトリックの近接グラフで同じノード（オブジェクト）を複数回探索する可能性があり，無駄な計算が行われてしまう．

1.3 本研究の貢献

本研究では，上記の課題を解決したアルゴリズムを提案する．提案アルゴリズムは，インデックス構築フェーズと探索フェーズに分けられる．インデックス構築フェーズでは，各メトリックにおいて，無向近似 K-Nearest Neighbor（KNN）グラフを構築し，各無向近似 KNN グラフ上で均等に分布した代表ノードを選定する．各メトリックでクエリに最も近い最短代表ノードを算出し，その隣接ノードとの最大距離を比較することで，初期探索ノードと探索メトリックの順序を決定する．あるメトリックでの探索を終了し，次のメトリックに移行する際，暫定解の中で現在のメトリックにおける近接グラフにおいて最もクエリに近いノードから探索を開始する．これを繰り返し，すべてのメトリックでの探索が終了するまで解の更新を行う．このアルゴリズムにより，クエリに近いかつ，クエリの周りにノードが密集している可能性が高い無向近似 KNN グラフの初期探索ノードから探索を行い，効率的に解の更新を行なえる．

また，初期探索ノード決定アルゴリズムとして，代表ノードに対して構築した VP 木 [19] を使用するアルゴリズムも提案する．探索フェーズでは，最も重みが大きいメトリックで構築された VP 木を用いて近似最近傍点を計算する．この近似最近傍点からグラフ探索を行い，暫定解を計算する．得られた暫定解を初期探索ノードとし，メトリックの探索順は重みの大きい順とする．このアルゴリズムにより，クエリとの近さと探索時間のトレードオフを考慮できる．本研究の主な貢献は以下の通りである．

- 探索の正確かつ素早い収束を目的とし，近接グラフに代表ノードおよび VP 木を導入する．
- 高次元空間における代表ノードおよび VP 木を使用した高速な近似マルチメトリック k-最近傍探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．

2 予備知識

マルチメトリックデータ． $O = \{o_1, \ldots, o_n\}$ を n 個のオブジェクトの集合とする．ただし，全ての $o_i \in O$ は m 個のメトリック空間を持ち，$o_i = \{o_i^1, \ldots, o_i^m\}$ と表す．$O^i = \{o_1^i, \ldots, o_n^i\}$ を i 番目のメトリックにおける n 個のオブジェクトの集合とする．$D = \{d_1(\cdot,\cdot), \ldots, d_m(\cdot,\cdot)\}$ を m 個のメトリックの集合とする．$\forall d_i(\cdot,\cdot) \in D$ は $\forall o_x^i, o_y^i, o_z^i \in O^i$ に対して，以下を満たす．

$$
\begin{aligned}
&d_i(o_x^i, o_y^i) = 0 \iff o_x^i = o_y^i \\
&d_i(o_x^i, o_y^i) = d_i(o_y^i, o_x^i) \\
&d_i(o_x^i, o_z^i) \le d_i(o_x^i, o_y^i) + d_i(o_y^i, o_z^i)
\end{aligned}
$$

問題定義． $W = \{w_1, \ldots, w_m\}$ を各メトリックに対応した m 個の重みの集合とする．各 $w_i \in W$ は $w_i \in [0, 1]$ であり，$\sum_{w_i \in W} w_i = 1$ である．このとき，m 個のメトリックからなる距離を以下に定義する．

定義 1（マルチメトリック距離）． $W = \{w_1, \ldots, w_m\}$ が与えられたとき，$\forall o_x, o_y \in O$ に対するマルチメトリック距離 $d^W(o_x, o_y)$ を以下のように定義する．ここで，$\overline{d_i}(\cdot,\cdot)$ は $d_i(\cdot,\cdot)$ を正規化した値であり，$\bar{d_i}(\cdot,\cdot) \in [0, 1]$ である．

$$
d^W(o_x, o_y) = \sum_{d_i \in D} w_i \cdot \overline{d_i}(o_x^i, o_y^i)
$$

この距離を用いて最近傍探索を定義する．

定義 2（マルチメトリック最近傍探索）． O，W，クエリ q，および出力サイズ k_s が与えられたとき，マルチメトリック最近傍探索は $S \subseteq O$，$|S| = k_s$，$\forall o_x \in S$，$\forall o_y \in O - S$ に対して，以下を満たす S を求める．

$$
d^W(o_x, q) \le d^W(o_y, q)
$$

近接グラフ． 各 $o^i \in O^i$ を i 番目のメトリック空間におけるノードとみなす．グラフ次数が k_g である近接グラフを $G_i = (O^i, E^i)$ とし，近接グラフの集合を $\mathcal{G} = \{G_1, \ldots, G_m\}$ とする．ここで E^i はエッジの集合であり，各 $o_x^i \in O^i$ のエッジ集合 $E^i[o_x^i]$ は，k_g 最近傍グラフの場合，o_x^i に最も近い k_s 個のノード $o_y^i \in O^i$ から成る．さらに，エッジの集合 E^i に対する逆エッジの集合を R^i とし，$R^i[o_x^i]$ は，$R^i[o_x^i] = \{o_y^i \in O^i | o_x^i \in E^i[o_y^i]\}$ である．

ベースラインアルゴリズム． マルチメトリック最近傍探索問題に対する近接グラフを用いた素朴なアルゴリズムを Algorithm 1に示す．このアルゴリズムでは，$w_i > 0$ の場合，初期探索ノード o_s を無作為に決定し，i 番目のメトリック空間の解集合 B^i と候補集合 B' に加える．B' の先頭のノード o_x を取り出し，o_x の隣接ノードの内，未探索であるノード $o_{x'}$ に対して探索を行う．$o_{x'}$ が B^i を更新する場合，B^i と B' に $o_{x'}$ を追加し，B^i の末尾のノードを削除する．これを B' が空集合になるまで繰り返す．この操作を重みが 0 より大きい全てのメトリックの近接グラフに対して行う．最後に，各グラフでの解をマージし，最終的な解を出力する．（Asure AI Search は本質的にこのアプロー

Algorithm 1: 近接グラフを用いた素朴な近似マルチメトリック k-最近傍探索

Input: $O, \mathcal{G}, q, k_s, W$, および候補容量パラメータ ϵ

$R \leftarrow \emptyset$
foreach $i \in [1, m]$ *s.t.* $w_i > 0$ **do**
  $B^i, B', F \leftarrow \emptyset$ (F is a set of visited nodes)
  $o_s \leftarrow$ a randomly selected start node
  Add $\langle o_s, d^W(o_s, q)\rangle$ into B^i and B' (B^i and B' are sorted by $d^W(\cdot, q)$)
  Add o_s into F
  while $B' \neq \emptyset$ **do**
    $o_x \leftarrow$ the top element in B'
    Erase $\langle o_x, d^W(o_x, q)\rangle$ from B'
    foreach $o_{x'} \in G_i(o_x)$ *s.t.* $o_{x'} \notin F$ **do**
      Add $o_{x'}$ into F
      $\tau \leftarrow$ the $\epsilon \cdot k_s$-th distance in B^i
      if $d^W(o_{x'}, q) < \tau$ **then**
        Add $\langle o_{x'}, d^W(o_{x'}, q)\rangle$ into B^i
        if $|B^i| > \epsilon \cdot k_s$ **then**
          Erase the $(k_s + 1)$-th element in B^i
        Add $\langle o_{x'}, d^W(o_{x'}, q)\rangle$ into B'
  Add top-k_s elements in B^i into R
return top-k_s elements in R

チを実装しており，各メトリックに対して独立的に実行するアイデアは人工知能分野の手法 [34] や近年のベクトル管理システム [31] でも利用されている．）

3 関連研究

シングルメトリック最近傍探索 は広く研究されており，様々なインデックスが提案されている．既存のインデックスは主にデータ空間を分割するコンパクトパーティショニング型 [15] および全てのオブジェクトの内，ピボットと呼ばれるオブジェクトを選択し，あらかじめすべてのオブジェクトまたは一部のオブジェクトに対して，ピボットとの距離を計算しておくピボット型 [5, 6, 10, 11] に分類される．さらに，コンパクトパーティショニング型とピボット型を組み合わせたハイブリッド型も存在する [12, 13]．シングルメトリック最近傍探索の多くの手法では，予め計算された距離に基づいてインデックス構築を行っている．しかし，マルチメトリック最近傍探索では任意の重みを考慮するため，あらかじめ距離を計算しておくことは困難である．したがって，既存のシングルメトリック最近傍探索アルゴリズムはマルチメトリック最近傍探索に適用できない．

マルチメトリック最近傍探索. 1章で述べたように，マルチメトリック最近傍探索を効率化するための既存のアプローチは,1 つのインデックスを構築する統合型 [8, 9, 14, 18] および各メトリック空間に対してインデックスを構築する分割型 [7, 28, 36] に分類される．

統合型のインデックスのほとんどは，各メトリックの重みを均等に考慮してインデックスを構築しており，重みを考慮できない．各メトリックごとに距離を計算しておき，探索時に重みを考慮する RR*-tree [18] も存在しているが，参照ノードの個数およびメトリックの個数が増加すれば，次元の呪いにより性能が大幅に低下する．また，統合型は全てのメトリックに対して単一のインデックスを構築するため，新しいメトリックの追加・削除に適していない．

分割型の各手法は，メトリック毎にシングルメトリック最近傍探索を行い，各メトリック毎の解をマージすることによって最終的な解を出力する．そのため，このアプローチの探索時間 T は各メトリックでの探索時間を T^i とすると，

$$T = \sum_m T^i + km \tag{1}$$

と表され，m に比例して増加する．ここで，m を削減するのは不可能であるため，T を削減するには T^i を削減しなければならない．T^i はどの種類のインデックスを用いるかによって決まり，既存研究の多くは近接グラフ以外のインデックスを用いている．しかし，シングルメトリック最近傍探索手法のベンチマーク [3] において，高次元空間における探索では，近接グラフ以外のインデックスは近接グラフに比べて性能が著しく劣ることが示されている．既存手法において高次元空間における探索は意識されておらず，高次元空間において高速な探索を行う手法は考案されていない．

4 提案アルゴリズム

最近傍探索では，距離計算回数の削減が探索時間の削減に直結する．先述したとおり，分割型では各メトリックの特徴を掴みやすく，重みを考慮することができるため，分割型を採用する．シングルメトリック最近傍探索では，近接グラフを用いた手法が最も精度と速度のトレードオフがとれている．そのため，各メトリックにおいて近接グラフを構築する．また，提案アルゴリズムでは 2 つ目以降のグラフで暫定解を使用し，クエリに近いノードから探索を始め，解を更新する．この過程において解の精度を上げるためには，クエリ周辺に存在するノードが探索される必要がある．そのためには，各ノード（オブジェクト）がそれに類似するノード（オブジェクト）と接続されていなければならない．したがって，提案アルゴリズムでは各メトリックで無向近似 k_gNN グラフを使用する．ここで，i 番目のメトリック空間における無向近似 k_gNN グラフを $U_i = (O^i, V^i)$ とし，これらの近接グラフの集合を $\mathcal{U} = \{U_1, \cdots, U_m\}$ とする．ここで，V^i は i 番目のメトリック空間における無向近似 k_gNN グラフのエッジの集合であり，有向近似 k_gNN グラフのエッジの集合 E^i および逆エッジの集合 R^i に対して，$V^i = E^i \cup R^i$ を満たす．

また，提案アルゴリズムでは探索順が 2 番目以降のメトリックにおける近接グラフにおいて暫定解から探索を始めることにより，2 番目以降のメトリックにおける探索時間は比較的短いことが期待できる．つまり，メトリック $\mathcal{M}_i$ の探索順を i としたとき，式（1）において，$T_1 \gg T_i$（$i \in [2, m]$）となる．これから分かるように，1 つ目のメトリックでは，暫定解の使用ができないため，このメトリックにおける探索がボトルネックとなってしまう．これを解消するためには，最初に探索するメトリックにおける近接グラフにおいて，クエリに近いノードを初期探索ノードとする必要がある．そこで，提案アルゴリズムでは 2 種類の初期探索ノード決定アルゴリズムを導入する．

1 つ目は代表ノードを用いたアルゴリズムである．各近接グラフ上で均等に分布した代表ノードを選出し，各代表ノードの k_g 番目の隣接ノードとの距離（隣接最大距離）を記録する．隣接最大距離が小さい代表ノードは，他の代表ノードと比較して，より近くに隣接ノードが存在している．したがって，クエリに近く，隣接最大距離が小さい代表ノードは，解になり得る可能性が高いノードが周辺に多く存在している．そのような代表ノードが存在する近接グラフにおいて，その代表ノードから探索することにより，解の収束速度を上げることを狙う．

2 つ目は代表ノードから構築した VP 木を用いたアルゴリズムである．各メトリック毎に選出した代表ノードをマ

ージし，それらに対して各メトリックごとに VP 木を構築する．ここでは重みが大きいメトリックのグラフから探索するものとし，最も重みが大きいメトリックで構築した VP 木に対して，複数回の乱択近似最近傍探索を行い，初期探索ノードを決定する．

4.1 インデックス構築

このフェーズはオフラインで行われ，探索フェーズにおけるパラメータに依存しない．そのため，静的なデータに対して一度のみ行われる．

無向近似 k_gNN グラフ． KNN グラフを構築する最も単純な方法は，各オブジェクトのペアの距離をすべて計算することである．しかし，この方法の計算量は $O(n^2)$ であり，大量データにスケールしない．したがって，効率的に高精度な KNN グラフを構築できる NNdescent[16] を使用する．このアルゴリズムを各メトリックに適用することにより，各メトリックにおける（つまり O^i の）近似 k_gNN グラフを効率的に構築する．ここで，近似 k_gNN グラフは有向グラフであることに注意する．近接グラフの接続性（到達可能性）を向上するため，このグラフを無向グラフにする [1]．

代表ノード． 提案アルゴリズムでは，効果的な初期探索ノードを導入するため，各メトリック空間で複数の代表ノードを選出する．ここで i 番目のメトリックにおける代表ノードの集合を $C^i = \{c^i_1, \ldots, c^i_{k_c}\}$ とし，$C = \{C^1, \ldots, C^m\}$ とする．探索時に全ての代表ノードとクエリとの距離を計算し，最もクエリに近い代表ノードから探索を開始することで探索範囲を削減する．しかし，代表ノードの位置が偏っている場合，任意のクエリに対して代表ノードの探索範囲削減効果を最大化できない．そのため，各近接グラフ上で均等に分布した代表ノードを選出する必要がある．そこで，k-means クラスタリング問題において高品質な初期クラスタ中心を選出するために考案された k-means++[2] を利用する．このアルゴリズムによって選出されたクラスタ中心を代表ノードとする．

また，各メトリックにおいて選出された代表ノードの隣接最大距離を記録する．i 番目のメトリックにおける隣接最大距離の集合を $MD^i = \{md^i_1, \ldots, md^i_{k_c}\}$ とし，$\mathcal{MD} = \{MD^1, \ldots, MD^m\}$ とする．ただし，各 $md^i \in MD^i$ は，対応する $c^i \in C^i$ から k_g 番目に近いノード $o^i \in O^i$ に対して，$md^i = \overline{d_i}(c^i, o^i)$ である．探索フェーズにおいてこの距離を考慮することにより，クエリ周辺にノードが密集している近接グラフから探索することを可能にする．ここで，各メトリック毎に距離のスケールが異なることに注意する．そのため，以下の z 正規化を行っている（$mean_i$ および std_i はそれぞれ i 番目のメトリックにおける距離の平均値および標準偏差である）．

$$\overline{d_i}(o^i_x, o^i_y) = \frac{d_i(o^i_x, o^i_y) - mean_i}{std_i}$$

VP 木． もう一つの初期探索ノード決定アルゴリズムとして VP 木を利用したものを提案する．VP 木を使用する目的は，短時間である程度クエリに近いノードを見つけることである．これを満たすため，各メトリック毎に選出した代表ノードの和集合 $C_{all} = \bigcup_{i=1}^{m} C^i$ に対して，VP 木を各メトリックに応じた距離関数で構築する．これにより，VP 木に格納するオブジェクト数を削減（木の探索時間を短縮）しつつ，クエリに近いノードを発見できる．

VP 木の各ノードはピボット vp，距離 th，左子ノードへのポインタ $left$，および右子ノードへのポインタ $right$ から成る．このアルゴリズムでは，無作為に初めの vp（根ノード）を決定し，vp と他のすべてのオブジェクトとの距離を計算する．根ノードとの距離がこれらの距離の中央値以下となるオブジェクトの集合を根ノードの左部分木に，それ以外のオブジェクトを右部分木に渡す．この操作を再帰的に行うことにより，空間を分割したインデックスである VP 木を構築する．

Algorithm 2: Determine-Start-Node-1

```
Input: C, D, MD, W, およびクエリ q
Output: o_s（初期探索ノード）and GO（探索メトリック順リスト）
for i = 1 to m do
    Add ⟨i, c^i, w_i · md_{c^i}⟩ into S s.t. min_{c^i ∈ C^i} d_i(c^i, q^i) (S is sorted by w_i · md_{c^i})
o_s ← S[0].c^i
foreach ⟨i, c^i, w_i · md_{c^i}⟩ ∈ S do
    Add i into GO
```

Algorithm 3: Determine-Start-Node-2

```
Input: q, W, d_i, and k_t（VP 木探索回数）
Output: o_s, GO
Add metric number in order of its weight into GO
root ← the root node of VP-tree in the most weighted metric
foreach i = 1 to k_t do
    bi ← ∅, bd ← ∞
    Search-VPtree(q, root, bi, bd, d_{GO[0]})
    Add ⟨bi, bd⟩ into S (S is sorted by the first element)
o_s ← the second element in the first structure in S
```

4.2 探索

提案アルゴリズムでは，初期探索グラフで得られた暫定解を利用して他メトリックにおける近接グラフの探索開始ノードを決定し，解を順次更新する．2 つ目以降のメトリックにおける近接グラフでは，暫定解を利用できるため，解の収束は早いことが期待できる．しかし，1 つ目のメトリックでは暫定解は利用できない．したがって，メトリックの探索順および初期探索ノードを適切に決定し，1 つ目のメトリックにおける探索時間を削減する必要がある．そこで，本論文では代表ノードおよび隣接最大距離を用いた探索メトリックの順序および初期探索ノード決定アルゴリズムを 2 つ提案する．

アプローチ ①． 代表ノードおよび代表ノードの隣接最大距離を利用し，探索メトリックの順序および初期探索ノードを決定するアルゴリズムを Algorithm 2に示す．各 $i \in [1, m]$ において，$\forall c^i \in C^i$ に対して $d_i(c^i, q^i)$ を計算する．その中で $d_i(c^i, q^i)$ が最も小さい c^i を $\langle i, c^i, w_i \cdot md_{c^i} \rangle$ として S に追加する．S は $w_i \cdot md_{c^i}$ によってソートされており，初期探索ノード o_s を S の先頭要素の代表ノードとする．S の先頭から要素 x を取りだし，x のグラフ番号（メトリックの識別子）をリスト GO に追加する．

アプローチ ②． 上のアプローチでは全ての代表ノードにアクセスする必要があるため，このコストの削減を VP 木を用いて実現する．C_{all} に対して構築した VP 木を利用し，探索メトリックの順序および初期探索ノードを決定するアルゴリズムを Algorithm 3に示す．本アルゴリズムでは，重みの大きさで探索メトリックの順序を決定する．そのため，全てのメトリックの VP 木を探索する必要はなく，最も重みが大きいメトリックの VP 木のみを探索する．

初期探索ノードの決定においては，初期探索ノード探索時間とクエリからの近さのトレードオフを考える必要

Algorithm 4: Search-VPtree

Input: q, u (node), bi, bd, d_i
1 **if** *u is a leaf node* **then**
2 　**return** $u.vp$
3 $\theta \leftarrow d_i(q, u.vp)$
4 **if** $\theta < bd$ **then**
5 　$bi \leftarrow u.vp$, $bd \leftarrow \theta$
6 **if** $dvp < u.th$ **then**
7 　**case** $u.th - bd \leq \theta < u.th + bd$ **do**
8 　　$rnd \leftarrow$ a random number $\in [0, 1]$
9 　　**if** $rnd < 0.5$ **then**
10 　　　Search-VPtree($q, u.left, bi, bd, d_i$)
11 　　**else**
12 　　　Search-VPtree($q, u.right, bi, bd, d_i$)
13 　**case** $(\theta < u.th + bd) \wedge (\theta < u.th - bd)$ **do**
14 　　Search-VPtree($q, u.left, bi, bd, d_i$)
15 **else**
16 　**case** $u.th - bd \leq \theta < u.th + bd$ **do**
17 　　Run lines 8–12
18 　**case** $(\theta \geq u.th + bd) \wedge (\theta \geq u.th - bd)$ **do**
19 　　Search-VPtree($q, u.right, bi, bd, d_i$)
20 **return** bd

がある．そのため，短時間でクエリに近いノード（オブジェクト）を見つけるために近似探索を複数回行う．このアルゴリズムを Algorithm 4に示す．VP 木の探索では，中心 vp および半径 th の円（円 ①）と中心 q および半径 bd の円（円 ②）の位置関係によってどちらかの部分木，もしくは両部分木を探索するか決定する．ここで，bi，bd，および θ はそれぞれ，現状見つかっているクエリから最も近いノード，bi と q の距離，および vp と q の距離である．初めに，θ と th で比較を行う．行6を満たす場合，vp を中心とする円内に q が存在する．このとき，行7を満たせば円 ① と円 ② が交差しており，両部分木に解が存在する可能性があることを意味する．本アルゴリズムではこの場合に，無作為に探索する部分木を決定する．行13を満たせば円 ② が円 ① に内包されていることを意味するため，左部分木のみを探索する．行6を満たさない場合，vp を中心とする円外に q が存在する．このとき，行16を満たせば円 ① と円 ② が交差しており，両部分木に解が存在する可能性があることを意味する．ここでも同様に無作為に探索する部分木を決定する．行18を満たせば円 ① の外部に円 ② が存在していることを意味するため，右部分木のみを探索する．これをある葉ノードにアクセスするまで繰り返す．この近似探索を k_t 回行い，最もクエリに近いオブジェクトを初期探索ノードとする．

近似 kNN 探索. 提案アルゴリズムを Algorithm 5に示す．初めに，初期探索ノード o_s およびメトリックの探索順序を決定する．その後，o_s を探索済みにする．

各 $x \in GO$ に対し，近接グラフ U_x の探索を行う．ただし，$w_x = 0$ のメトリックでは探索を行わない．U_x が初めて探索するメトリックにおける近接グラフである場合，初期探索ノード o_s とクエリ q の距離を計算し，B（暫定解の集合）および B'（探索ノードの候補）に追加する．そうでない場合，暫定解の中で q に最も近いノードを初期探索ノード $o_{s'}$ にする．次に，$o_{s'}$ と q の距離を計算し，B' に追加する．B' の先頭ノードを探索ノード o_y とし，o_y を B' から削除する．o_y の隣接ノードの内，未探索のノード $o_{y'}$ を探索済みにし，τ に B の中で $\epsilon \cdot k_s$ 番目の距離を記録する．$o_{y'}$ とクエリとの距離計算を行い，その値が τ 未満であれ

Algorithm 5: 近似マルチメトリック k-最近傍探索

Input: O, $\mathcal{U}$, q, ϵ, and k_s
$(o_s, GO) \leftarrow$ Determine-Start-Node-1 or Determine-Start-Node-2
Add o_s into F (F is visit check)
$B \leftarrow \emptyset$
foreach $x \in GO$ **do**
　if $w_x \neq 0$ **then**
　　$B' \leftarrow \emptyset$
　　if $B = \emptyset$ **then**
　　　Add $\langle o_s, d^W(o_s, q)\rangle$ into B and B' (B and B' are sorted by $d^W(\cdot, q)$)
　　else
　　　$o_{s'} \leftarrow \arg\min_{o \in B} d_x(o^x, q^x)$
　　　Add $\langle o_{s'}, d^W(o_{s'}, q)\rangle$ into B'
　　while $B' \neq \emptyset$ **do**
　　　$o_y \leftarrow$ the top element in B'
　　　Erase $\langle o_y, d^W(o_y, q)\rangle$ from B'
　　　foreach $o_{y'} \in U_x(o_y)$ *s.t.* $o_{y'} \notin F$ **do**
　　　　Add $o_{y'}$ into F
　　　　$\tau \leftarrow$ the $\epsilon \cdot k_s$-th distance in B
　　　　if $d^W(o_{y'}, q) < \tau$ **then**
　　　　　Add $\langle o_{y'}, d^W(o_{y'}, q)\rangle$ into B and B'
　　　　　if $|B| > \epsilon \cdot k_s$ **then**
　　　　　　Erase the $(k_s + 1)$-th element in B
return the first k_s elements in B

ば，$o_{y'}$ を B および B' に追加する．B の要素数が容量 $\epsilon \cdot k_s$ を超えれば，$k_s + 1$ 番目の要素を削除する．上の操作を B' が空になるまで繰り返す．B' が空になれば反復を終了し，次のグラフでの探索を行う．これを決定したメトリックの探索順で実行する．

5 評価実験

5.1 設定

データセット. 評価実験において，2 つの実データセットを使用した．使用したデータセットの詳細を表1に示す．Corel Image Features[3]は Color histogram（CH），Layout histogram（LH），Color moments（CM），および Cooc texture（CT）で構成される．評価実験において，CH および LH に L_1 距離を使用し，CM および CT に L_2 距離を使用した．NUS-WIDE[4]は, CH, Color correlogram（CORR), Edge direction histogram（EDH），Wavelet texture（WT），および Block-wise color moments extracted over 5x5 fixed grid partitions（CM55）で構成される．評価実験において，CH, CORR, および EDH に L_1 距離を使用し，WT および CM55 に L_2 距離を使用した．そして，それぞれのデータセットから無作為に 1,000 個のデータを選択し，それらをクエリ q とした．

パラメータ. 評価実験では様々なパラメータを変化させ，提案アルゴリズムの性能を評価した．インデックス構築および探索に使用するパラメータの詳細を表2に示す．k_g，k_c，k_s，および k_t はそれぞれ，有向 k_gNN グラフの次数，代表ノードの個数，探索における解の個数，および VP 木における近似探索の回数である．Corel Image Features および NUS-WIDE では，各メトリックにおける代表ノードの個数

[3]https://kdd.ics.uci.edu/databases/CorelFeatures/CorelFeatures.html
[4]https://lms.comp.nus.edu.sg/wp-content/uploads/2019/research/nuswide/NUS-WIDE.html

表 1: データセットの詳細

データセット	データ数	次元数	メトリック数	距離関数
Corel Image Features	66,616	9 ~ 32	4	L_1 距離，L_2 距離
NUS-WIDE	269,648	64 ~ 228	5	L_1 距離，L_2 距離

を 1,000 個に設定した．重みの集合 W は $w_i \in W$ の合計が 1 になるよう無作為に決定した．

表 2: パラメータの詳細

パラメータ	値
k_g	20
k_c	1,000
k_s	10, 20, 100
k_t（Corel Image Features）	10, 50, 100
k_t（NUS-WIDE）	100, 150, 200

比較アルゴリズム. 提案アルゴリズムの性能を評価するため，Algorithm 1（Naive）との比較を行った．

実験環境および実装. 評価実験に使用するアルゴリズムはすべて C++ で実装し，評価実験における距離計算はすべて Single Instruction, Multiple Data streams（SIMD）命令を用いて最適化した．すべての評価実験は，Intel Core i9-9980XE 3.0GHz CPU，128GB の RAM，および Ubuntu 22.04.3 OS を搭載した計算機で行った．

評価方法. 評価実験では，1,000 回のクエリを発行したときの平均探索時間に対する 1,000 回のクエリを発行したときの平均リコールで比較を行った．ここで，各アルゴリズムから得られた解集合を X，線形探索によって得られた正しい解集合を Y としたとき，アルゴリズムから得られた解のリコール $Recall(X)$ を以下のように定義する．

$$Recall(X) = \frac{|X \cap Y|}{|Y|}$$

5.2 結果

本節では，評価実験の結果を示し考察する．ここで，グラフ探索順および初期探索ノード決定アルゴリズム ① を使用した提案アルゴリズムを提案アルゴリズム ①（Ours①），メトリック探索順および初期探索ノード決定アルゴリズム ② を使用した提案アルゴリズムを提案アルゴリズム ②（Ours②）とする．比較資料として，各データセットにおいて線形探索を行った場合の平均探索時間を表3に示す．

Corel Image Features. 図3に k_s = 10，20，および 100 とした場合の，Corel Image Features における提案アルゴリズム ①，提案アルゴリズム ②（k_t = 10，50，および 100），および Naive の結果を示す．本データセットにおいては，k_s = 10，20，および 100 すべての場合で提案アルゴリズム ① および提案アルゴリズム ② が Naive を上回り，約 4 倍（メトリック数倍）高速である．また，表3と比較して，提案アルゴリズムは高精度な解を 10 倍以上高速に出力できていることが分かる．提案アルゴリズム ② の k_t の違いによる性能差はほとんど見られない．しかし，k_s が増加するにつれて，提案アルゴリズム ② の提案アルゴリズム ① に対する優位性が減少し，k_s = 100 では提案アルゴリズム ① が提案アルゴリズム ② を上回っている．

表 3: 各データセットにおける線形探索時間

データセット	平均探索時間 [ms]
Corel Image Features	32.28
NUS-WIDE	227.69

NUS-WIDE. 図4に k_s = 10，20，および 100 とした場合の，NUS-WIDE における提案アルゴリズム ①，提案アルゴリズム ②（k_t = 100，150，および 200），および Naive の結果を示す．本データセットにおいても，k_s = 10，20，および 100 すべての場合で提案アルゴリズム ① および提案アルゴリズム ② が Naive を上回り，約 5 倍高速である．Corel Image Features での結果と同様に，提案アルゴリズム ② の k_t の違いによる性能差はほとんど見られない．一方，Corel Image Features とは違い，提案アルゴリズム ① と提案アルゴリズム ② の探索性能差もほとんど見られない．

解の共有. 表4および表5に同一平均 recall 帯における各メトリックでの平均探索時間を示す．これらの表から分かるように，Naive では，各メトリックでほとんど同じ探索時間がかかっているのに対し，提案アルゴリズムでは 2 つ目以降のメトリックでの探索ではほとんど探索時間がかかっていない．そのため，両データセットにおいてメトリック数倍の高速化を実現できており，これは解の共有の有効性を示している．

代表ノードおよび VP 木. 提案アルゴリズム ① では，代表ノードの集合に対して線形探索を行っているが，提案アルゴリズム ② では，代表ノードの集合の VP 木に対して近似探索を行っている．表4および表5から分かるように提案アルゴリズム ① は，提案アルゴリズム ② よりも初期探索ノードの探索にかかる時間は大きいが，よりクエリに近いノードから探索を始めることができるため，$\mathcal{M}_1$ における探索時間が提案アルゴリズム ② よりも短くなっている．

提案アルゴリズムのグラフ探索において，あるメトリックにおける必要な距離計算回数は “$\epsilon \times k_s \times$ ホップ数” と概算できる．k_s の値が小さい場合，初期探索ノードが遠くなり，ホップ数が増えることによる総探索時間に与える影響は比較的小さい．しかし，k_s の値が大きくなるにつれて，初期探索ノードが遠くなり，ホップ数が増えることによる総探索時間に与える影響は大きくなる．したがって，k_s が小さい場合，提案アルゴリズム ② の初期探索ノードの探索にかかる時間が短いという利点が，初期探索ノードが提案アルゴリズム ① と比較してクエリから遠いという欠点を上回るため，提案アルゴリズム ② が提案アルゴリズム ① よりも早い．しかし，k_s が大きくなるにつれて，利点と欠点の影響が逆転していくため，最終的に提案アルゴリズム ① が提案アルゴリズム ② を上回っている．そのため，表4の k_s = 10 では，提案アルゴリズム ① よりも提案アルゴリズム ② の方が，“初期探索ノード探索にかかる時間” + “初めに探索されたメトリックでの探索時間” が短いが，k_s = 100 では同じになっている．また，データ数およびメトリック数が増加すれば，初期探索グラフにかかる探索時間の相対的な影響は減るため，NUS-WIDE における提案アルゴリズム ① と提案アルゴリズム ② の差がほとんどないと考えられる．

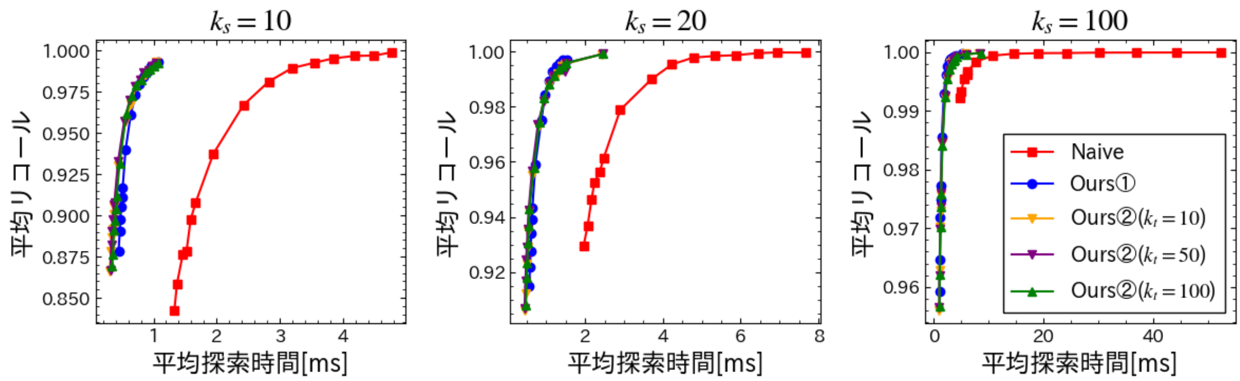

図 3: Corel Image Features における結果

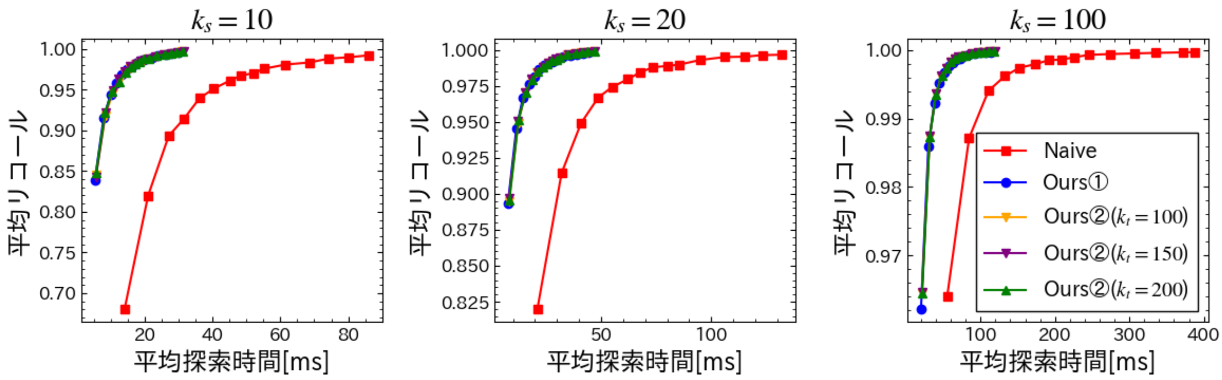

図 4: NUS-WIDE における結果

表 4: 同一平均 recall 帯における各メトリックでの平均探索時間 [ms]（Corel Image Features）．$\mathcal{M}_i$ は i 番目に探索されたメトリックを示す．

k_s	手法	初期探索ノード	$\mathcal{M}_1$	$\mathcal{M}_2$	$\mathcal{M}_3$	$\mathcal{M}_4$	Recall
10	Naive	-	0.413	0.397	0.388	0.229	0.901
	Ours①	0.215	0.237	0.013	0.009	0.009	0.905
	Ours②（$k_t = 10$）	0.017	0.306	0.028	0.011	0.009	0.907
	Ours②（$k_t = 50$）	0.060	0.261	0.026	0.011	0.009	0.905
	Ours②（$k_t = 100$）	0.096	0.249	0.026	0.011	0.009	0.903
100	Ours①	0.227	0.777	0.022	0.013	0.012	0.959
	Ours②（$k_t = 10$）	0.019	0.992	0.070	0.015	0.012	0.962
	Ours②（$k_t = 50$）	0.067	0.941	0.070	0.015	0.012	0.962
	Ours②（$k_t = 100$）	0.106	0.927	0.071	0.016	0.012	0.963

6　まとめ

本研究では，マルチメトリック k-最近傍探索問題に取り組んだ．素朴なアルゴリズムでは，各メトリックごとに構築したインデックスを利用した探索を独立に行って解をマージするため，探索時間が長くなる．そのため，提案アルゴリズムではインデックス構築フェーズで各メトリックごとに近接グラフを構築し，隣接最大距離を保持した代表ノードの設置および VP 木の構築を行った．探索フェーズでそれらを活用し，最適なメトリック探索順および初期探索ノードを決定した．また，全メトリックを考慮した重み付き距離で探索を行うことにより，各メトリックでの解の共有を可能にし，探索範囲の削減を実現した．実データでの評価実験により，提案アルゴリズムの性能が素朴なアルゴリズムの性能を大きく上回ることを確認した．

謝辞

本研究の一部は，AIP 加速課題（JPMJCR23U2）の支援を受けたものである．

参考文献

[1] Yusuke Arai, Daichi Amagata, Sumio Fujita, and Takahiro Hara. 2021. LGTM: A Fast and Accurate kNN Search Algorithm in High-dimensional Spaces. In *DEXA*. 220–231.

[2] David Arthur and Sergei Vassilvitskii. 2007. K-means++ the Advantages of Careful Seeding. In *SODA*. 1027–1035.

表 5: 同一平均 recall 帯における各メトリックでの平均探索時間 [ms]（NUS-WIDE）．$\mathcal{M}_i$ は i 番目に探索されたメトリックを示す．

k_s	手法	初期探索ノード	$\mathcal{M}_1$	$\mathcal{M}_2$	$\mathcal{M}_3$	$\mathcal{M}_4$	$\mathcal{M}_5$	Recall
10	Naive	-	5.635	5.096	5.838	6.538	4.162	0.900
	Ours①	0.554	6.027	0.189	0.118	0.084	0.077	0.899
	Ours②（$k_t = 10$）	0.234	6.466	0.194	0.098	0.077	0.060	0.900
	Ours②（$k_t = 50$）	0.321	6.596	0.192	0.099	0.076	0.062	0.907
	Ours②（$k_t = 100$）	0.399	6.367	0.181	0.097	0.078	0.061	0.900
100	Ours①	0.622	18.099	0.474	0.257	0.180	0.167	0.960
	Ours②（$k_t = 10$）	0.250	18.740	0.448	0.224	0.164	0.129	0.960
	Ours②（$k_t = 50$）	0.346	18.666	0.449	0.228	0.164	0.128	0.960
	Ours②（$k_t = 100$）	0.436	18.865	0.459	0.223	0.162	0.129	0.959

[3] Martin Aumüller, Erik Bernhardsson, and Alexander Faithfull. 2017. Ann-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. In *SISAP*. 34–49.

[4] Artem Babenko and Victor Lempitsky. 2016. Efficient Indexing of Billion-scale Datasets of Deep Descriptors. In *CVPR*. 2055–2063.

[5] Tolga Bozkaya and Meral Ozsoyoglu. 1997. Distance-based Indexing for High-dimensional Metric Spaces. In *SIGMOD*. 357–368.

[6] Tolga Bozkaya and Z. Meral Özsoyoglu. 1999. Indexing Large Metric Spaces for Similarity Search Queries. *ACM Transactions on Database Systems (TODS)* 24, 3 (1999), 361–404.

[7] Benjamin Bustos, Daniel Keim, and Tobias Schreck. 2005. A Pivot-based Index Structure for Combination of Feature Vectors. In *SAC*. 1180–1184.

[8] Benjamin Bustos, Sebastian Kreft, and Tomáš Skopal. 2012. Adapting metric indexes for searching in multi-metric spaces. *Multimedia Tools and Applications* 58, 3 (2012), 467–496.

[9] Benjamin Bustos and Tomáš Skopal. 2006. Dynamic Similarity Search in Multi-metric Spaces. In *MIR*. 137–146.

[10] Lu Chen, Yunjun Gao, Xinhan Li, Christian S. Jensen, and Gang Chen. 2015. Efficient Metric Indexing for Similarity Search. In *ICDE*. 591–602.

[11] Lu Chen, Yunjun Gao, Xinhan Li, Christian S. Jensen, and Gang Chen. 2017. Efficient Metric Indexing for Similarity Search and Similarity Joins. *IEEE Transactions on Knowledge and Data Engineering* 29, 3 (2017), 556–571.

[12] Lu Chen, Yunjun Gao, Xuan Song, Zheng Li, Yifan Zhu, Xiaoye Miao, and Christian S. Jensen. 2023. Indexing Metric Spaces for Exact Similarity Search. *ACM Computing Survey* 55, 6 (2023), 128:1–128:39.

[13] Lu Chen, Yunjun Gao, Baihua Zheng, Christian S. Jensen, Hanyu Yang, and Keyu Yang. 2017. Pivot-based Metric Indexing. *Proceedings of the VLDB Endowment* 10, 10 (2017), 1058–1069.

[14] Paolo Ciaccia and Marco Patella. 2000. The M2-tree: Processing Complex Multi-Feature Queries with Just One Index. In *DELOS*.

[15] Paolo Ciaccia, Marco Patella, Pavel Zezula, et al. 1997. M-tree: An efficient access method for similarity search in metric spaces. In *VLDB*, Vol. 97. 426–435.

[16] Wei Dong, Charikar Moses, and Kai Li. 2011. Efficient K-nearest Neighbor Graph Construction for Generic Similarity Measures. In *WWW*. 577–586.

[17] Yury Elkin and Vitaliy Kurlin. 2021. A new compressed cover tree guarantees a near linear parameterized complexity for all k-nearest neighbors search in metric spaces. *CoRR* abs/2111.15478 (2021).

[18] Maximilian Franzke, Tobias Emrich, Andreas Züfle, and Matthias Renz. 2016. Indexing Multi-metric Data. In *ICDE*. 1122–1133.

[19] Ada Wai-chee Fu, Polly Mei-shuen Chan, Yin-Ling Cheung, and Yiu Sang Moon. 2000. Dynamic vp-tree indexing for n-nearest neighbor search given pair-wise distances. *The VLDB Journal* 9 (2000), 154–173.

[20] Jinyang Gao, Hosagrahar Visvesvaraya Jagadish, Wei Lu, and Beng Chin Ooi. 2014. DSH: data sensitive hashing for high-dimensional k-nnsearch. In *In SIGMOD*. 1127–1138.

[21] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. 2013. Optimized Product Quantization. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36, 4 (2013), 744–755.

[22] Ben Harwood and Tom Drummond. 2016. FANNG: Fast Approximate Nearest Neighbour Graphs. In *CVPR*. 5713–5722.

[23] Herve Jegou, Matthijs Douze, and Cordelia Schmid. 2010. Product Quantization for Nearest Neighbor Search. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33, 1 (2010), 117–128.

[24] Wen Li, Ying Zhang, Yifang Sun, Wei Wang, Mingjie Li, Wenjie Zhang, and Xuemin Lin. 2020. Approximate Nearest Neighbor Search on High Dimensional Data—Experiments, Analyses, and Improvement. *IEEE Transactions on Knowledge and Data Engineering* 32, 8 (2020), 1475–1488.

[25] Kejing Lu, Mineichi Kudo, Chuan Xiao, and Yoshiharu Ishikawa. 2021. HVS: Hierarchical Graph Structure based on Voronoi Diagrams for Solving Approximate Nearest Neighbor Search. *Proceedings of the VLDB Endowment* 15, 2 (2021), 246–258.

[26] Yu A Malkov and Dmitry A Yashunin. 2018. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence* 42, 4 (2018), 824–836.

[27] Yongjoo Park, Michael Cafarella, and Barzan Mozafari. 2015. Neighbor-sensitive Hashing. *Proceedings of the VLDB Endowment* 9, 3 (2015), 144–155.

[28] Kostas Patroumpas and Dimitrios Skoutas. 2020. Similarity Search over Enriched Geospatial Data. In *GeoRich*. 1:1–1:6.

[29] Parikshit Ram, Dongryeol Lee, William March, and Alexander Gray. 2009. Linear-time Algorithms for Pairwise statistical problems. *In NIPS* 22 (2009), 1527–1535.

[30] Shulong Tan, Zhixin Zhou, Zhaozhuo Xu, and Ping Li. 2020. Fast Item Ranking under Neural Network Based Measures. In *WSDM*. 591–599.

[31] Jianguo Wang, Xiaomeng Yi, Rentong Guo, Hai Jin, Peng Xu, Shengjun Li, Xiangyu Wang, Xiangzhou Guo, Chengming Li, Xiaohai Xu, et al. 2021. Milvus: A Purpose-Built Vector Data Management System. In *SIGMOD*. 2614–2627.

[32] Jiahui Yu, Zirui Wang, Vijay Vasudevan, Legg Yeung, Mojtaba Seyedhosseini, and Yonghui Wu. 2022. Coca: Contrastive Captioners are Image-text Foundation Models. *arXiv preprint arXiv:2205.01917* (2022).

[33] Yuxiang Zeng, Yongxin Tong, and Lei Chen. 2021. Hst+: An efficient index for embedding arbitrary metric spaces. In *ICDE*. 648–659.

[34] Shaoting Zhang, Ming Yang, Timothee Cour, Kai Yu, and Dimitris N Metaxas. 2014. Query Specific Rank Fusion for Image Retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37, 4 (2014), 803–815.

[35] Weijie Zhao, Shulong Tan, and Ping Li. 2020. Song: Approximate Nearest Neighbor Search on GPU. In *ICDE*. 1033–1044.

[36] Yifan Zhu, Lu Chen, Yunjun Gao, Baihua Zheng, and Pengfei Wang. 2022. DESIRE: An efficient dynamic cluster-based forest indexing for similarity search in multi-metric spaces. *Proceedings of the VLDB Endowment* 15, 10 (2022), 2121–2133.