DEIM2022 J24-2

高速かつ高精度な内積空間におけるカーディナリティ推定

平田　皓平† 　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: †{hirata.kohei,amagata.daichi,hara}@ist.osaka-u.ac.jp

あらまし　本論文は，内積空間におけるカーディナリティ推定問題を扱う．この問題は，高次元ベクトルの集合，クエリ，および閾値が与えられたときに，クエリとの内積が閾値以下となるベクトルの数を推定するものである．この問題は，ユーザやアイテムなどのオブジェクトを行列を用いて管理する最近の機械学習アプリケーションにとって重要である．例えば，この問題では，与えられたクエリ（アイテム）の市場規模，すなわち，そのクエリについて評価が肯定的であるユーザの数を推定する．この問題を解決するための重要な要件は，高い効率性と精度である．これらの要件を満たすために，我々はサンプリングベースのアルゴリズムを提案する．このアルゴリズムは，オフラインでユークリッド空間への変換と次元削減を行い，変換したベクトル集合を木構造を用いてインデクシングする．そして，このインデックスにおいて検索範囲と交差する葉ノードに存在するベクトルをサンプリングする．我々のアルゴリズムは，理論的および実践的に高速で高精度である．実世界のデータを用いた実験により，提案アルゴリズムが既存の技術と比較して優れた性能であることを示した．

キーワード　カーディナリティ推定，内積空間

1　はじめに

近年，人工知能の技術が実世界のアプリケーションに応用されている．代表的な例として，深層学習や行列分解などが挙げられる．これらの技術では，ニューロン，ユーザ，アイテムなどのオブジェクトを行列を用いて管理するため，Web [23]，統計学 [12]，機械学習 [24]，データベース [16,27]，および推薦システム [2,3] など，様々な分野で内積空間に着目した研究が行われている．その中でも，内積の探索問題は，ニューラルネットワークモデルや行列積から出力を抽出するための基本的な演算であるため，広く研究されている [1,4,8,9,11,15,17,18,20–22,26,33,34,36]．本論文では，探索問題の推定版，すなわち，内積空間におけるカーディナリティ推定問題を扱う．この問題は，高次元ベクトルの集合 $\mathbf{X}$，クエリ $\mathbf{q}$，および閾値 τ が与えられたとき，$|\mathbf{X_q}|$ を推定するものである．$\mathbf{X_q}$ は，$\mathbf{x} \cdot \mathbf{q} \geqq \tau$ を満たす $\mathbf{x} \in \mathbf{X}$ からなるベクトル集合であり，$\mathbf{x} \cdot \mathbf{q}$ は $\mathbf{x}$ と $\mathbf{q}$ の内積である．この問題には，統計的密度推定 [32]，ジョイン処理スケジュールの作成 [26]，推薦システムにおける市場規模の把握 [2] などの重要な応用例がある．

この問題を解く素朴な方法は，最新の内積探索アルゴリズムを実行することである．この方法では正確なカーディナリティを得るが，計算量が大きいという問題がある．これは，最新の探索アルゴリズムが $\mathbf{X}$ の線形探索に基づいており [16,27]，$n = |\mathbf{X}|$ の場合，時間計算量が $O(n)$ となるためである．ここで，多くの近似内積探索アルゴリズムが研究されている [15,20,22,33]．これらはカーディナリティ推定の高速化のために活用することができる．しかし，これらのアルゴリズムは $\alpha < 1$ とし，$O(n^\alpha + |\mathbf{X_q}|)$ の時間計算量が必要である．つまり，依然として多くのベクトルにアクセスする必要があるため，この方法は不適切である．別の方法として，内積空間が変換可能な他のデータ空間に対する推定アルゴリズムを用いることである．例えば，ベクトルを角距離空間へ変換した時，角距離におけるカーディナリティ推定方法を用いることができる [32]．しかし，これはチューニングが困難なパラメータを多く持つため，実践的でない．さらに，我々の評価実験ではこの方法は誤差が大きいことを示している．

以上から，内積のカーディナリティ推定問題は挑戦的であり，既存技術では，高速かつ高精度な推定をすることが不可能である．そこで我々は，内積空間におけるカーディナリティ推定のための新しいアルゴリズムを提案する．我々のアイデアは，この問題を低次元ユークリッド空間における近似範囲カウンティング問題に変換することである．このアイデアに基づいて，我々のアルゴリズムは，高精度なカーディナリティ推定に十分な局所空間からのみベクトルをサンプリングする．この方法により，少ないサンプル数で高精度なカーディナリティが得られ，高速かつ高精度な推定が可能となる．我々の主な貢献は以下である．

- 内積空間におけるカーディナリティ推定のための新しいアルゴリズムを提案する．
- 提案アルゴリズムの性能を理論的に分析する．
- 実世界のデータセットを用いて大規模な実験を行う．実験の結果，我々のアルゴリズムは，最新のカーディナリティ推定アルゴリズムよりも高精度にカーディナリティを推定する．また，我々のアルゴリズムは，推定誤差を小さく抑えつつ，最新の内積探索アルゴリズムよりもはるかに高速であることを示す．

2 問題定義

$\mathbf{X}$ は n 個のデータベクトル $\mathbf{x}_1, ..., \mathbf{x}_n$ からなるデータ集合とする．各データベクトル $\mathbf{x} \in \mathbf{X}$ は d 次元であり，$\mathbf{x} = \langle \mathbf{x}[1], ..., \mathbf{x}[d] \rangle$ である．また，d は大きいと仮定する．2 つのベクトル $\mathbf{x}$ および $\mathbf{x}'$ が与えられた時，内積 $\mathbf{x} \cdot \mathbf{x}'$ は以下で求まる．

$$\mathbf{x} \cdot \mathbf{x}' = \sum_{i=1}^{n} \mathbf{x}[i] \cdot \mathbf{x}'[i]$$

通常，内積を用いるアプリケーションは，与えられたクエリベクトル $\mathbf{q}$ との内積が大きくなるベクトルに関心がある．例えば，$\mathbf{q}$ が新しいアイテムを表す場合，$\mathbf{q}$ との内積（例えば，アイテムに対する評価）が大きいユーザベクトルを検索するアプリケーションが存在する．このようなベクトルの数を推定するために，以下の問題に取り組む．

定義 1 (内積空間におけるカーディナリティ推定)．データベクトルの集合 $\mathbf{X}$，クエリベクトル $\mathbf{q}$，および閾値 τ が与えられた時，集合 $\mathbf{X_q}$ を次のように定義する．

$$\mathbf{X_q} = \{\mathbf{x} | \mathbf{x} \in \mathbf{X}, \mathbf{x} \cdot \mathbf{q} \geqq \tau\}$$

ここで，内積空間におけるカーディナリティ推定問題は，$|\mathbf{X_q}|$ を推定することである．

我々の目的は，任意のクエリ $\mathbf{q}$ に対して，$|\mathbf{X_q}|$ を高速かつ高精度に推定することである．

3 関連研究

カーディナリティ推定は多くのアプリケーションにとって必要不可欠な演算であるため，多くの研究で高速かつ高精度なカーディナリティ推定技術が提案されている．その典型的な例が，スケッチによる集合のカーディナリティ推定である [13]．有名なスケッチ技術に Count-Min スケッチ [7] や HyperLogLog スケッチ [10] がある．また，これらの変種は [28] などで広く研究されている．しかし，これらは多次元のベクトルには利用できない．

データベース分野では，多次元空間におけるカーディナリティ推定問題を扱った研究が多く存在する．古典的な解決方法は，$\mathbf{X}$ から単純にランダムサンプリングを行うものである．実際には，この単純なランダムサンプリングでは精度が高くない．他の方法として，カーネル密度推定 [19] に基づいたものがある．しかし，内積がメトリックに従わないのに対し，この方法はメトリック空間のみに対応している [1]．近年，カーディナリティ推定のための機械学習モデルも研究されている [25, 29–31]．これらの研究は，リレーショナルデータベースのテーブル [31] やメトリック空間 [25, 30] を対象としたものである．したがって，これらの方法は本問題に適用するのが困難であり，内積空間に特有な方法が必要であることを表している．さらに，我々は内積のカーディナリティ推定のための機械学習モデルは考慮しない．これは，[25] などの既存研究においてカーディナリティ推定モデルの学習に数時間以上の長い時間が必要であることが示されているからである．(一方，我々の前処理は数分以下で済む．)

本問題には，[32] で提案されているカーディナリティ推定アルゴリズムを適用することができる．実際，内積探索は角距離を用いた類似検索に変換することができる ([3] が内積探索をコサイン類似度における類似検索に変換できることを示しており，2 つのベクトルのコサイン類似度から 2 点間の角距離が得られる)．このアルゴリズムは，locality-sesitive hashing(LSH) [5] を利用し，各ベクトルのハッシュ値を計算する．LSH の各バケットには，同じハッシュ値を持つベクトルが保持される．クエリベクトルが与えられた時，まずクエリベクトルのハッシュ値を計算する．そして，クエリのハッシュ値とハミング距離が閾値以下となるバケットからサンプリングを行う．ただし，閾値はハミング距離におけるものである．このアルゴリズムでは，各サンプルベクトルとクエリとの角距離を計算し，サンプリング結果からカーディナリティを推定する．しかし，推定結果の精度に直接関係する LSH のパラメータやハミング距離における閾値は明らかではなく，このアルゴリズムは実用的ではない．

内積探索． 本問題を解決するための簡単な方法は，内積探索アルゴリズムを実行することである．厳密な探索アルゴリズムには，空間分割 [21] と線形探索 [16, 27] の 2 つの方法がある．空間分割に基づく検索アルゴリズム [21] は，高次元ベクトルのインデックスにコーン木を利用する．次元の呪いにより，高次元ベクトルに対する空間分割に基づくアルゴリズムは，線形探索に基づくアルゴリズムよりも性能が低い．このことから，最新の厳密な探索アルゴリズム [16, 27] は，線形探索を採用し，探索の早期終了技術を提案している．しかし，この方法は時間計算量が $O(n)$ であり，カーディナリティ推定には適切でない．

高次元ベクトルの場合，厳密な探索アルゴリズムは計算コストが高いため，近似探索アルゴリズムが考案されている．典型的な方法は LSH [15, 20, 22, 23] を利用することである．この方法は，近似内積探索の問題を近似最近傍検索に変換するものである．また，近接グラフ [17, 34, 36] を利用する方法も存在する．この方法では貪欲法を使用し，解の更新が無くなるまで，与えられたクエリに最も近い近接グラフのノードを走査する．これらの方法とは異なり，最新の近似内積探索アルゴリズムである ScaNN [11] はベクトル量子化を採用している．本質的には，ベクトル量子化の考え方は LSH と似ており，データ空間を互いに独立した部分空間に分割する．そして，与えられた $\mathbf{q}$ について対応する部分空間のみを探索する．ScaNN は，ベクトル量子化によって生じる損失を最小限に抑えながら，ベクトル量子化を行う．しかし，この最新の方法でさえ，多くのベクトルにアクセスする必要があるため，カーディナリティ推定には非効率的である．

1：4.1 節で，我々はユークリッド変換を用いるが，これはベクトルとクエリの間でのみ成立し，$\mathbf{X}$ の任意の 2 つのベクトル間では成立しない．

4 提案アルゴリズム

望ましい解決方法は，部分集合 $\mathbf{X}' \in \mathbf{X}$ にのみアクセスすることである．ただし，各 $\mathbf{x} \in \mathbf{X}'$ は $\mathbf{x} \cdot \mathbf{q} \geqq \tau$ を満たす．しかし，τ を事前に知ることができないため，この方法を実現するのは困難である．そこで，これを近似的に実現する方法について詳細を説明する．

低次元ユークリッド空間では，木構造を用いた範囲検索は多くの不要な点を枝刈りできるため，有望な手法である．そこで我々は，内積空間からユークリッド空間への変換および高次元空間から低次元空間への変換という 2 つの変換を行う．なお，単純に範囲検索を行う場合，時間計算量が $\Omega(|\mathbf{X_q}|)$ であり，$|\mathbf{X_q}|$ が大きい場合があるため，効率性が低い可能性がある．この問題を避けるために，サンプリングによる方法を考える．しかし，このような木構造上の範囲検索において，高精度な推定を行うために，どのようにサンプリングを取り入れるかは明らかではない．この課題に取り組み，理論的および実践的に高速かつ高精度なアルゴリズムを提案する．

4.1 オフラインアルゴリズム

最初に，提案アルゴリズムのデータ構造を構築するオフラインアルゴリズムについて説明する．なお，このオフライン処理は一度しか行われず，構築されたデータ構造は任意の $\mathbf{q}$ および τ に利用できる．

(1) ノルムで $\mathbf{X}$ をソート．まず，各 $\mathbf{x} \in \mathbf{X}$ について L2 ノルムを計算する．そして，このノルムの降順で $\mathbf{X}$ をソートする．

(2) 内積空間からユークリッド空間への変換．この変換を実現するために，Xbox transformation [3] を使用する．これは，各 $\mathbf{x} \in \mathbf{X}$ を次のように $\bar{\mathbf{x}}$ に変換する．

$$\bar{\mathbf{x}} = \langle \mathbf{x}[1], ..., \mathbf{x}[d], \sqrt{M^2 - ||\mathbf{x}||^2} \rangle$$

ここで，M は $\mathbf{X}$ の中の最大ノルムである．また，与えられたクエリ $\mathbf{q}$ を次のように $\bar{\mathbf{q}}$ に変換する．

$$\bar{\mathbf{q}} = \langle \mathbf{q}[1], ..., \mathbf{q}[d], 0 \rangle \tag{1}$$

ここで，

$$\begin{aligned} ||\bar{\mathbf{x}} - \bar{\mathbf{q}}||^2 &= ||\mathbf{x} - \mathbf{q}||^2 + M^2 - ||\mathbf{x}||^2 \\ &= M^2 + ||\mathbf{q}||^2 - 2\mathbf{x} \cdot \mathbf{q} \end{aligned}$$

であり，以下が成り立つ．

$$\mathbf{x} \cdot \mathbf{q} \geqq \tau \Leftrightarrow ||\bar{\mathbf{x}} - \bar{\mathbf{q}}||^2 \leqq M^2 + ||\mathbf{q}||^2 - 2\tau.$$

r を次のように定義する．

$$r = \sqrt{M^2 + ||\mathbf{q}||^2 - 2\tau}.$$

$|\mathbf{X_q}|$ は $||\bar{\mathbf{x}} - \bar{\mathbf{q}}|| \leqq r$ を満たす $\bar{\mathbf{x}}$ の数に対応する．以上より，内積のカーディナリティ推定問題は，ユークリッド空間における範囲検索に変換できることがわかる．変換されたベクトルの集合を $\bar{\mathbf{X}}$ とする．

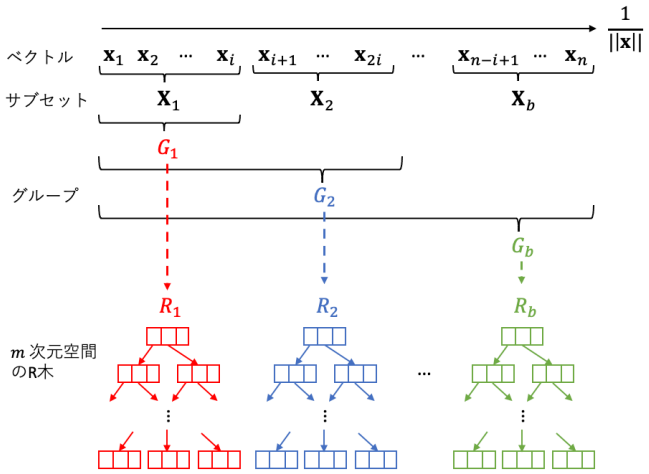

図 1: データ構造の概要．$||\mathbf{x}_1|| \geqq ||\mathbf{x}_2|| \geqq \cdots \geqq ||\mathbf{x}_n||$ と仮定する．$\mathbf{X}$ は独立したサブセット $\mathbf{X}_1, \mathbf{X}_2, ..., \mathbf{X}_b$ に分割される．各グループ G_i は $\mathbf{X}_1, ..., \mathbf{X}_i$ を含む．各グループ G_i に対し，m 次元空間において R 木 R_i を構築する．

(3) 次元削減．高次元ユークリッド空間における範囲検索問題は効率的に解くことができないため，$\bar{\mathbf{X}}$ の次元を $(d+1)$ 次元から m 次元まで削減する．このとき，次のランダムな写像 $h(\bar{\mathbf{x}})$ を使用する．

$$h(\bar{\mathbf{x}}) = a \cdot \bar{\mathbf{x}}.$$

ここで，a は $(d+1)$ 次元のベクトルであり，その各次元は正規分布 $N(0,1)$ に従うランダムな値である．$\tilde{\mathbf{x}}$ は写像空間における $\mathbf{x}$ とする．すなわち，

$$\tilde{\mathbf{x}} = \langle h_1(\bar{\mathbf{x}}), ..., h_m(\bar{\mathbf{x}}) \rangle. \tag{2}$$

(4) m 次元空間における木構造の作成．ノルムの順序に基づいて，$\mathbf{X}$ を b 個の独立したサブセット $\mathbf{X}_1, ..., \mathbf{X}_b$ に均等に分割する．ここで，$\bigcup_{i=1}^{b} \mathbf{X}_i = \mathbf{X}$ および $\mathbf{X}_i \cap \mathbf{X}_j = \varnothing$ $(i \neq j)$ であり，$b = O(\log n)$ とする．次に，b 個のグループ $G_1, ..., G_b$ を作成する．グループ G_i には $\mathbf{X}_1, ...\mathbf{X}_i$ が含まれる (すなわち，$G_i \subseteq G_{i+1}$)．最後に，各グループ G_i において，$\mathbf{x} \in \bigcup_{j=1}^{i} \mathbf{X}_j$ に対する写像後のベクトル $\tilde{\mathbf{x}}$ からなる集合について，R 木 R_i を構築する．図 1 にデータ構造を示す．

$\mathbf{X}_i$ における最大ノルムを M_i とする．コーシーシュワルツの不等式，すなわち，$\mathbf{x} \cdot \mathbf{q} \leqq ||\mathbf{x}||||\mathbf{q}||$ から，$M_i||\mathbf{q}|| < \tau$ の場合，サブセット $\mathbf{X}_i$ をフィルタリングする．この場合，全ての $\mathbf{x} \in \bigcup_{j=i}^{b} \mathbf{X}_j$ は $\mathbf{x} \cdot \mathbf{q} \geqq \tau$ を満たさないからである．例として，図 1 において，$M_3||\mathbf{q}|| < \tau$ の場合，R_2 のみを使用する．これは各サブセット $\mathbf{X}_i$（各グループではない）に対して，R 木を構築することでも実現可能である．しかし，その場合，複数の R 木（つまり，$\mathbf{X}_1$ の R 木から $\mathbf{X}_{i-1}$ の R 木）を探索しなければならず，多くのノード探索コストがかかる．グループを作成することでこの欠点を回避しており，これが提案アルゴリズムのデータ構造の背景にある考えである．

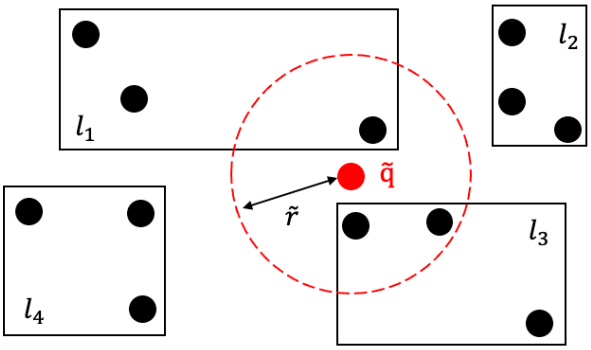

図 2: 範囲検索の例．写像後のクエリ $\tilde{\mathbf{q}}$ は赤の点で示されており，点線の円は $\tilde{\mathbf{q}}$ を中心とする検索範囲を示している．R 木の葉ノード l_1, l_2, l_3 および l_4 は長方形で表され，黒の点は写像後のベクトルを示す．この範囲検索の結果は $\{l_1, l_3\}$ となる．

4.2 オンラインアルゴリズム

次に，このデータ構造を利用し，高速かつ高精度にカーディナリティを推定するオンラインアルゴリズムについて説明する．
(1) 不要なベクトルのフィルタリング．クエリ $\mathbf{q}$ および閾値 τ が与えられたとき，まず，クエリのノルム $||\mathbf{q}||$ を計算する．次に，4.1 節で述べたように，コーシーシュワルツの不等式から不要なベクトルをフィルタリングする．つまり，グループ G_i を得る．

$$i = \max_{1 \leqq j \leqq b} j \quad \text{s.t.} \quad M_j ||\mathbf{q}|| \geqq \tau$$

であり，$\mathbf{X}_{i+j} (j \geqq 1)$ はフィルタリングされる．
(2) R 木探索．検索範囲と交差する R 木のノードを検索する．クエリ $\mathbf{q}$ を式 (1) により $\tilde{\mathbf{q}}$ に変換し，式 (2) により m 次元空間に写像し，$\tilde{\mathbf{q}}$ を得る．これに伴い，$\tilde{r} = \sqrt{m} r$ を計算する．ここで，$r = \sqrt{M^2 + ||\mathbf{q}||^2 - 2\tau}$ である．($\tilde{r}$ の設定については 4.3 節で説明する．) 次に，R_i 上で範囲検索を行う．なお，点を検索する通常の範囲検索とは異なり，我々の範囲検索では，$\tilde{\mathbf{q}}$ を中心とする半径 $\tilde{r}$ の超球と交差する R_i の葉ノードを検索する．例として，図 2 では範囲検索により，葉ノード l_1 および l_3 が検索される．
(3) サンプルからのカーディナリティ推定．最後に，上記で得られた葉ノードを用いて，カーディナリティ $|\mathbf{X}_q|$ を推定する．$\mathbf{V}_{\tilde{\mathbf{q}},\tilde{r}}$ を範囲検索で得られた葉ノードに属する写像後のベクトルからなる集合とする．$\mathbf{V}_{\tilde{\mathbf{q}},\tilde{r}}$ においてランダムサンプリングを行う．(サンプルサイズを s とし，$|\mathbf{V}_{\tilde{\mathbf{q}},\tilde{r}}| \gg s$ と仮定している．もし，$|\mathbf{V}_{\tilde{\mathbf{q}},\tilde{r}}| \leqq s$ の場合，単純に $\mathbf{V}_{\tilde{\mathbf{q}},\tilde{r}}$ をスキャンする．)
具体的には，カウンタ c を準備し，次のように行う．

(i) $\mathbf{V}_{\tilde{\mathbf{q}},\tilde{r}}$ から写像後のベクトルをランダムサンプリングする．

(ii) サンプリングされたベクトルを $\tilde{\mathbf{x}}$ とし，$\mathbf{x} \cdot \mathbf{q}$ を計算する．

(iii) $\mathbf{x} \cdot \mathbf{q} \geqq \tau$ の場合，c をインクリメントする．

(iv) (i)〜(iii) を s 回繰り返す．

最後に，$|\mathbf{X}_\mathbf{q}|$ の推定値として，$\frac{c \cdot |\mathbf{V}_{\tilde{\mathbf{q}},\tilde{r}}|}{s}$ を使用する．

4.3 分　　析

簡単化のため，$d = O(1)$ とする．まず，我々のデータ構造の空間計算量について説明する．

定理 1 (空間計算量)．我々のデータ構造は，$O(n \log n)$ の空間を必要とする．

証明. $b = O(\log n)$ 個の R 木があり，$i \in [1, b-1]$ に対し，$G_i \subseteq G_{i+1}$ であるグループ G_i がある．したがって，$\mathbf{x}$ の写像後のベクトル $\tilde{\mathbf{x}}$ は最大で $O(\log n)$ 回複製される．この事実から，この定理は成立する． □

次に，提案アルゴリズムの時間計算量について説明する．

定理 2 (時間計算量)．提案アルゴリズムにおけるオンラインの時間計算量は，$O(\log n + N + s)$ である．ただし，N は超球と交差する R_i の葉ノードの数である．

証明. $b = O(\log n)$ 個のサブセットがあるため，フィルタリングのステップは最大でも $O(\log n)$ の時間計算量を必要とする．超球と交差する R_i の葉ノードの数を N とし，R 木を探索するステップでは，$O(\log n + N)$ の時間計算量を必要とする．最後に，ランダムサンプリングには $O(s)$ の時間計算量が必要であり，$|\mathbf{V}_{\tilde{\mathbf{q}},\tilde{r}}|$ を求めるのに $O(N)$ の時間計算量が必要である．つまり，提案アルゴリズムは $O(\log n + N + s)$ の時間計算量が必要である． □

次に，我々のオンラインアルゴリズムの有効性を分析する．まず最初に，2 つのベクトルのユークリッド距離を $dist(\cdot, \cdot)$ と表し，任意の 2 つのベクトル $\mathbf{x_1}$ および $\mathbf{x_2}$ に対し，$\bar{t} = dist(\bar{\mathbf{x_1}}, \bar{\mathbf{x_2}})$ とする．また，$\tilde{t} = dist(\tilde{\mathbf{x_1}}, \tilde{\mathbf{x_2}})$ とする．次に，以下の補助定理を紹介する [35].

補助定理 1 $t = \frac{\tilde{t}}{\sqrt{m}}$ とする．このとき，$\mathbb{E}[\tilde{t}] = \sqrt{m} \times \bar{t}$ であり，t は $\bar{t}$ の不偏推定量である．

この補助定理から，$||\bar{\mathbf{x}} - \bar{\mathbf{q}}|| \leqq r$ を満たす $\bar{\mathbf{x}}$ は $||\tilde{\mathbf{x}} - \tilde{\mathbf{q}}|| \leqq \tilde{r} = \sqrt{m} r$ であることが期待できる．ランダムサンプリングの性質上，次のようになる．

系 1 $\mathbb{E}\left[\frac{c \cdot |\mathbf{V}_{\tilde{\mathbf{q}},\tilde{r}}|}{s}\right] = |\mathbf{X}_\mathbf{q}|$.

さらに，$\mathbf{x} \cdot \mathbf{q} \geqq \tau$ を満たす $\mathbf{x}$ に対応する $\tilde{\mathbf{x}}$ をサンプリングする確率 p を求めることができる．

$$p = \frac{|\mathbf{X}_\mathbf{q}|}{|\mathbf{V}_{\tilde{\mathbf{q}},\tilde{r}}|}. \tag{3}$$

これを基に，我々の推定の分散を導く．

補助定理 2 $\mathrm{Var}\left[\frac{c \cdot |\mathbf{V}_{\tilde{\mathbf{q}},\tilde{r}}|}{s}\right] = \frac{|\mathbf{X}_\mathbf{q}|(|\mathbf{V}_{\tilde{\mathbf{q}},\tilde{r}}| - |\mathbf{X}_\mathbf{q}|)}{s}$.

証明. 独立した確率変数 X_i について考える．これは，確率 p で $X_i = 1$，確率 $1-p$ で $X_i = 0$ となる．$X = \sum_{i=1}^{s} X_i$ とする．(これは，我々のアルゴリズムの c に対応する．) $\mathrm{Var}[X_i] = p(1-p)$

および $\mathrm{Var}[X] = sp(1-p)$ であるので，X を $\frac{\mathrm{c}\cdot|\mathbf{V}_{\tilde{\mathbf{q}},\tilde{r}}|}{s}$ に置き換えることにより，

$$\mathrm{Var}\left[\frac{\mathrm{c}\cdot|\mathbf{V}_{\tilde{\mathbf{q}},\tilde{r}}|}{\mathrm{s}}\right] = \left(\frac{|\mathbf{V}_{\tilde{\mathbf{q}},\tilde{r}}|}{s}\right)^2 \mathrm{Var}[\mathrm{c}] = \frac{|\mathbf{V}_{\tilde{\mathbf{q}},\tilde{r}}|}{s}^2 p(1-p).$$

式 (3) から，この補助定理は成立する． □

この補助定理は，(i) サンプル数が多いほど推定の分散が小さくなること，および (ii)$p \approx \frac{1}{2}$ でない限り分散は小さくなることを示している．最後に，次の補助定理について説明する．

補助定理 3 我々のアルゴリズムは，$s = O\left(\frac{|\mathbf{V}_{\tilde{\mathbf{q}},\tilde{r}}|}{|\mathbf{X}_{\mathbf{q}}|\epsilon^2}\ln\frac{1}{\delta}\right)$ とすることで，少なくとも $1-\delta$ の確率で $(1\pm\epsilon)|\mathbf{X}_{\mathbf{q}}|$ となる推定値を返す．

証明. 補助定理 2 と同じ表記法を使用する．なお，$\mathbb{E}[X] = sp$. $0<\epsilon<1$，$0<\delta<1$ に対し，$\Pr[|X-\mathbb{E}[X]| \geqq \epsilon\mathbb{E}[X]] \leqq \delta$ を満たす s が必要である．チェルノフ限界から，

$$\Pr[|\mathrm{X}-\mathbb{E}[\mathrm{X}]| \geqq \epsilon\mathbb{E}[\mathrm{X}]] \leqq 2\mathrm{e}^{-\frac{\mathbb{E}[\mathrm{X}]\epsilon^2}{3}} = 2\mathrm{e}^{-\frac{sp\epsilon^2}{3}}.$$

それゆえ，

$$\delta \leqq 2e^{-\frac{sp\epsilon^2}{3}}$$

となる必要がある．これより，以下のようになる．

$$s \geqq \frac{3}{p}\left(\frac{1}{\delta}\right)^2 \ln\left(\frac{1}{\delta}\right) = O\left(\frac{1}{p\epsilon^2}\ln\left(\frac{1}{\delta}\right)\right).$$

式 (3) から，

$$s = O\left(\frac{|\mathbf{V}_{\tilde{\mathbf{q}},\tilde{r}}|}{|\mathbf{X}_{\mathbf{q}}|\epsilon^2}\ln\frac{1}{\delta}\right).$$

従って，この定理が成立する． □

我々のアルゴリズムは，単純である一方で高速であり（定理 2），推定の期待値は正確であり（系 1），誤差は理論的に制御可能である (補助定理 2 および補助定理 3) ことが明らかである．また，コーシーシュワルツの不等式に基づくフィルタリングにより，サンプリングの複雑さが改善されている．

5 評価実験

この章では，実験結果について説明する．全ての実験は，CPU に 3.0GHz の Core i9-9980XE および 128GB の RAM を搭載した計算機上で行った．

5.1 設　　定

3 つの実世界のデータセット：Amazon-M(movie)，Amazon-K(Kindle) [14]，および Netflix[2]を使用した．これらは，レーティングデータの集合であり，レーティングの尺度は 1 から 5 である．ユーザおよびアイテムのベクトルを生成するために，[6] の Matrix Factorization を実行した．ユーザとアイテムの数を表 1 に示す．評価では，$|\mathbf{X}_{\mathbf{q}}| \geqq 1$ を満たすアイテムベクトルをクエリとし，ユーザベクトルの集合を $\mathbf{X}$ として設定した．

表 1: データセットの統計

	Amazon-M	Amazon-K	Netfilx
ユーザ数	2,088,620	1,406,890	480,189
アイテム数	200,941	430,530,	17,770

本実験では，以下のアルゴリズムについて評価を行った．

- FEXIPRO [16]：内積探索のための最新の厳密アルゴリズム．
- ScaNN [11]：k-最大内積探索のための最新の近似アルゴリズム．このアルゴリズムでは，τ を設定することができないため，$k = |\mathbf{X}_{\mathbf{q}}|$ とする．(これは実用的ではないが，ScaNN の最適な設定でその性能を知ることができる．)
- IS [32]：角距離におけるカーディナリティ推定のための最新のアルゴリズム．
- Facetts[3]：4 章で述べた我々の提案アルゴリズム．$m = 5$ に設定した．
- Facetts-wop：Facetts から $\mathbf{X}$ を分割する手順を除いたアルゴリズム．(そのため，コーシーシュワルツの不等式は使用できず，単一の R 木を使用する．)

ScaNN については，公開された実装 [4]を使用した．その他は，C++で実装を行い，g++ 5.5.0 に最適化オプション-O3 付加して実行した．

5.2 オフライン処理の結果

まず，Facetts のオフライン時間を測定した．Amazon-M, Amazon-K，および Netflix では，オフライン時間はそれぞれ 66.5 秒，43.9 秒，および 14.1 秒であった．Facetts のオフライン時間は短く，$|\mathbf{X}|$ に対して線形にスケールする．

5.3 オンライン処理の結果

(i) カーディナリティを推定するための平均時間，および (ii) 推定誤差を測定した．誤差の測定には，平均絶対パーセント誤差（APE）を用いた．また，

$$\mathrm{APE} = \left|\frac{|\mathbf{X}_{\mathbf{q}}| - C_{est}}{|\mathbf{X}_{\mathbf{q}}|}\right|$$

であり，C_{est} は推定したカーディナリティである．ScaNN の場合，得られた結果の中で閾値を満たすベクトルの数を C_{est} とする．また，C_{est} の平均値 (APE_{avg}) および中央値（APE_{med}）を求めた．

サンプル数の影響． 推定時間および誤差に及ぼすサンプル数 s の影響を調べた．$\tau = 4.0$ に設定した．実験結果を表 2，3，および 4 に示す（太文字は近似アルゴリズムの中で最も誤差が小

2：https://www.cs.uic.edu/liub/Netflix-KDD-Cup-2007.html

3：Fast and accurate cardinality estimation via tree traversal sampling

4：https://github.com/google-research/google-research/tree/master/scann

表 2: Amazon-M における APE の平均，APE の中央値，および実行時間 [msec] に対するサンプル数の影響

サンプル数	2000			4000			6000		
	APE_{avg}	APE_{med}	実行時間	APE_{avg}	APE_{med}	実行時間	APE_{avg}	APE_{med}	実行時間
IS	0.988	0.902	1.7	0.980	0.891	2.8	0.976	0.884	4.0
Facetts-wop	0.141	0.018	3.0	0.145	0.015	4.2	0.113	0.019	5.3
Facetts	**0.083**	**0.012**	3.1	**0.117**	**0.011**	4.3	**0.083**	**0.008**	5.4

表 3: Amazon-K における APE の平均，APE の中央値，および実行時間 [msec] に対するサンプル数の影響

サンプル数	2000			4000			6000		
	APE_{avg}	APE_{med}	実行時間	APE_{avg}	APE_{med}	実行時間	APE_{avg}	APE_{med}	実行時間
IS	0.901	0.855	1.7	0.892	0.846	2.8	0.891	0.842	3.9
Facetts-wop	0.126	0.023	2.3	0.110	0.016	3.4	0.100	0.017	4.5
Facetts	**0.081**	**0.017**	2.3	**0.070**	**0.013**	3.4	**0.060**	**0.011**	4.5

表 4: Nefflix における APE の平均，APE の中央値，および実行時間 [msec] に対するサンプル数の影響

サンプル数	2000			4000			6000		
	APE_{avg}	APE_{med}	実行時間	APE_{avg}	APE_{med}	実行時間	APE_{avg}	APE_{med}	実行時間
IS	0.440	0.357	1.0	0.421	0.348	1.8	0.410	0.342	2.6
Facetts-wop	0.209	0.061	1.2	0.181	0.047	2.2	0.141	0.039	3.1
Facetts	**0.116**	**0.036**	1.0	**0.091**	**0.026**	1.9	**0.076**	**0.026**	2.7

表 5: FEXIPRO および ScaNN における APE の平均，APE の中央値，および実行時間 [msec]

データセット	Amazon-M			Amazon-K			Netflix		
	APE_{avg}	APE_{med}	実行時間	APE_{avg}	APE_{med}	実行時間	APE_{avg}	APE_{med}	実行時間
FEXIPRO	0	0	521.8	0	0	325.8	0	0	52.0
ScaNN	0.106	0.047	81.0	0.105	0.052	50.0	0.187	0.118	4.6

表 6: Facetts の各操作にかかる時間 [msec]($s = 2000$)

データセット	Amazon-M	Amazon_K	Netflix
フィルタリング	0.005	0.006	0.005
木の探索	1.353	0.811	0.159
サンプリング	1.721	1.467	0.889

さいことを表す)．FEXIPRO および ScaNN はサンプリングを用いないため，$\tau = 4.0$ のときの実行時間を測定し，その結果を表 5 に示す．

一般的に，サンプル数が増加する場合，各サンプリングベースのアルゴリズムは，実行時間が長くなる一方で，より誤差は小さくなる．これは，理論的に当然の結果である．ここで，2 つの重要な結果が見られる．まず，Facetts はすべてのデータセットにおいて，IS よりもはるかに小さな誤差で推定を行なっている．例えば，Amazon-M と Amazon-K において，Facetts の APE_{avg} は，同程度の実行時間で，IS の約 10 倍小さい．(時間差は木の探索から生じており，IS は木の探索を用いていないためである．) この結果は，我々の手法が有効であることを示しており，他のデータ空間におけるカーディナリティ推定アルゴリズムを利用するだけでは有効ではないことを示している．2 つ目に，Facetts はサンプル数が少ない場合 (例えば $s = 2000$) においても誤差が十分に小さいのに対して，IS は大きな誤差が生じている．APE_{med} に注目すると，Facetts は少なくとも半分のクエリに対して 5 %以下の誤差で推定を行なっている．

次に，検索に基づいた最新のアルゴリズムである FEXIPRO および ScaNN と Facetts の比較を行う．Facetts は，正確な最新のアルゴリズムである FEXIPRO よりもはるかに高速である．また，Facetts は ScaNN よりも高速である．表 6 が示すように，Facetts の各操作は大きな計算コストを発生させないため，推定時間は非常に短い．なお，$s = 2000$ の場合においても，Facetts は ScaNN よりも小さい誤差で推定を行う．この結果から，検索ベースのアルゴリズムはカーディナリティ推定には適していないことが確認できる．

我々の分割アプローチの有効性を評価するために，Facetts とその変種である Facetts-wop との比較を行う．表 2 – 4 は，常に Facetts がより小さな誤差で推定を達成することを示している．その理由は，不要なベクトルをフィルタリングすることに由来する．m 次元空間への写像により，超球と交差する葉ノードにも $\mathbf{x} \cdot \mathbf{q} < \tau$ となる $\mathbf{x}$ が存在する．Facetts は，このような場合を可能な限り避けているため，推定誤差が小さくなっている．

閾値 τ の影響． 表 7 – 9 は，τ を変化させた場合の実験結果を示している．サンプリングに基づいたアルゴリズムは $s = 2000$ に設定した．τ が大きくなるにつれ，FEXIPRO および ScaNN の実行時間は短くなる．これは，閾値が大きくなるにつれて，探索空間が小さくなるためである．一方，サンプリングに基づ

表 7: Amazon-M における APE の平均，APE の中央値，および実行時間 [msec] に対する閾値の影響

τ	3.5			4			4.5		
	APE_{avg}	APE_{med}	実行時間	APE_{avg}	APE_{med}	実行時間	APE_{avg}	APE_{med}	実行時間
FEXIPRO	0	0	674.0	0	0	521.8	0	0	327.5
ScaNN	0.058	0.012	120.7	0.106	0.047	81.0	0.187	0.123	45.4
IS	1.063	0.961	1.8	0.988	0.902	1.7	0.861	0.766	1.5
Facetts-wop	0.074	0.009	3.1	0.141	0.018	3.0	0.244	0.046	2.8
Facetts	0.061	0.006	3.2	0.083	0.012	3.1	0.147	0.032	2.4

表 8: Amazon-K における APE の平均，APE の中央値，および実行時間 [msec] に対する閾値の影響

τ	3.5			4			4.5		
	APE_{avg}	APE_{med}	実行時間	APE_{avg}	APE_{med}	実行時間	APE_{avg}	APE_{med}	実行時間
FEXIPRO	0	0	432.9	0	0	325.8	0	0	192.2
ScaNN	0.056	0.018	79.1	0.105	0.052	50.0	0.191	0.157	23.9
IS	0.969	0.909	1.8	0.901	0.855	1.7	0.769	0.714	1.5
Facetts-wop	0.091	0.013	2.3	0.126	0.023	2.3	0.216	0.052	2.2
Facetts	0.059	0.011	2.3	0.081	0.017	2.3	0.134	0.035	1.9

表 9: Nefflix における APE の平均，APE の中央値，および実行時間 [msec] に対する閾値の影響

τ	3.5			4			4.5		
	APE_{avg}	APE_{med}	実行時間	APE_{avg}	APE_{med}	実行時間	APE_{avg}	APE_{med}	実行時間
FEXIPRO	0	0	116.5	0	0	52.0	0	0	15.1
ScaNN	0.097	0.069	12.5	0.187	0.118	4.6	0.348	0.234	1.4
IS	0.497	0.463	1.2	0.440	0.357	1.0	0.525	0.363	0.9
Facetts-wop	0.109	0.035	1.3	0.209	0.061	1.2	0.432	0.155	1.1
Facetts	0.067	0.026	1.3	0.116	0.036	1.0	0.293	0.074	0.7

いたアルゴリズムは，表 6 が示すように，サンプリングされたベクトルと与えられたクエリベクトルの内積を計算することが主な計算コストであるので，実行時間は大きく変化しない．

次に，τ が大きくなるにつれ，ScaNN およびサンプリングに基づいたアルゴリズムの誤差が大きくなることについて考察する．τ が大きい場合，$|\mathbf{X_q}|$ は減少する．このとき，$\mathbf{x}\cdot\mathbf{q} \geqq \tau$ を満たすベクトル $\mathbf{x}$ をサンプリングする確率が下がる．そのため，サンプリングに基づいたアルゴリズムの誤差は増加する．しかし，τ が大きい場合でも，Facetts は小さな誤差を維持している．特に，APE_{avg} は小さい．また，τ が小さい場合，ScaNN は Facetts より APE_{avg} が小さくなるが，その差は僅かであり，Facetts の方が非常に推定時間は短い．

6 結　　論

本論文では，内積空間におけるカーディナリティ推定問題を扱った．機械学習の技術は広く普及しており，高次元ベクトルの内積を使用することが多いため，この問題は重要なアプリケーションを持つ．(近似) 内積探索や他のデータ空間におけるカーディナリティ推定の既存技術を本問題を解決するために用いることできる．しかし，これらの技術は多くのデータアクセスを必要とするため，本問題において大きなコストがかかる．

そこで我々は，内積空間におけるカーディナリティを高速かつ高精度に推定するアルゴリズムを提案した．このアルゴリズムは，内積のカーディナリティ推定を低次元ユークリッド空間における範囲カウント推定に変換するという新しい考えを採用している．我々のアルゴリズムの性能を理論的に解析した結果，性能保証が可能である．また，提案アルゴリズムが実世界のデータセットにおいて高速かつ高精度なカーディナリティ推定を行うことを実証した．

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 JSTCREST(JPMJCR21F2) の支援を受けたものである．

文　　献

[1] F. Abuzaid, G. Sethi, P. Bailis, and M. Zaharia, “To index or not to index: Optimizing exact maximum inner product search,” In ICDE, pp.1250–1261, 2019.

[2] D. Amagata and T. Hara, “Reverse maximum inner product search: How to efficiently find users who would like to buy my item?,” In RecSys, pp.273–281, 2021.

[3] Y. Bachrach, Y. Finkelstein, R. Gilad-Bachrach, L. Katzir, N. Koenigstein, N. Nice, and U. Paquet, “Speeding up the xbox recommender system using a euclidean transformation for inner-product spaces,” In RecSys, pp.257–264, 2014.

[4] G. Ballard, T.G. Kolda, A. Pinar, and C. Seshadhri, “Diamond sampling for approximate maximum all-pairs dot-product (mad) search,” In ICDM, pp.11–20, 2015.

[5] M.S. Charikar, "Similarity estimation techniques from rounding algorithms," In STOC,, pp.380–388, 2002.

[6] W.S. Chin, B.W. Yuan, M.Y. Yang, Y. Zhuang, Y.C. Juan, and C.J. Lin, "Libmf: A library for parallel matrix factorization in shared-memory systems.," Journal Machine Learning Research, vol.17, no.1, pp.2971–2975, 2016.

[7] G. Cormode and S. Muthukrishnan, "An improved data stream summary: the count-min sketch and its applications," Journal of Algorithms, vol.55, no.1, pp.58–75, 2005.

[8] X. Dai, X. Yan, K.K. Ng, J. Liu, and J. Cheng, "Norm-explicit quantization: Improving vector quantization for maximum inner product search," In AAAI, pp.51–58, 2020.

[9] Q. Ding, H.F. Yu, and C.J. Hsieh, "A fast sampling algorithm for maximum inner product search," In AISTATS, pp.3004–3012, 2019.

[10] P. Flajolet, É. Fusy, O. Gandouet, and F. Meunier, "Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm," In Discrete Mathematics and Theoretical Computer Science, pp.137–156, 2007.

[11] R. Guo, P. Sun, E. Lindgren, Q. Geng, D. Simcha, F. Chern, and S. Kumar, "Accelerating large-scale inference with anisotropic vector quantization," In ICML, pp.3887–3896, 2020.

[12] I. Han and J. Gillenwater, "Map inference for customized determinantal point processes via maximum inner product search," In AISTATS, pp.2797–2807, 2020.

[13] H. Harmouch and F. Naumann, "Cardinality estimation: An experimental survey," PVLDB, vol.11, no.4, pp.499–512, 2017.

[14] R. He and J. McAuley, "Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering," In World Wide Web, pp.507–517, 2016.

[15] Q. Huang, G. Ma, J. Feng, Q. Fang, and A.K. Tung, "Accurate and fast asymmetric locality-sensitive hashing scheme for maximum inner product search," In SIGKDD, pp.1561–1570, 2018.

[16] H. Li, T.N. Chan, M.L. Yiu, and N. Mamoulis, "Fexipro: fast and exact inner product retrieval in recommender systems," In SIGMOD, pp.835–850, 2017.

[17] J. Liu, X. Yan, X. Dai, Z. Li, J. Cheng, and M.C. Yang, "Understanding and improving proximity graph based maximum inner product search," In AAAI, pp.139–146, 2020.

[18] R. Liu, T. Wu, and B. Mozafari, "A bandit approach to maximum inner product search," In AAAI, pp.4376–4383, 2019.

[19] M. Mattig, T. Fober, C. Beilschmidt, and B. Seeger, "Kernel-based cardinality estimation on metric data," In EDBT, pp.349–360, 2018.

[20] B. Neyshabur and N. Srebro, "On symmetric and asymmetric lshs for inner product search," In ICML, pp.1926–1934, 2015.

[21] P. Ram and A.G. Gray, "Maximum inner-product search using cone trees," In SIGKDD, pp.931–939, 2012.

[22] A. Shrivastava and P. Li, "Asymmetric lsh (alsh) for sublinear time maximum inner product search (mips)," NIPS, pp.2321–2329, 2014.

[23] A. Shrivastava and P. Li, "Asymmetric minwise hashing for indexing binary inner products and set containment," In World Wide Web, pp.981–991, 2015.

[24] R. Spring and A. Shrivastava, "Scalable and sustainable deep learning via randomized hashing," In SIGKDD, pp.445–454, 2017.

[25] J. Sun, G. Li, and N. Tang, "Learned cardinality estimation for similarity queries," In SIGMOD, pp.1745–1757, 2021.

[26] C. Teflioudi and R. Gemulla, "Exact and approximate maximum inner product search with lemp," ACM Transactions on Database Systems, vol.42, no.1, pp.1–49, 2017.

[27] C. Teflioudi, R. Gemulla, and O. Mykytiuk, "Lemp: Fast retrieval of large entries in a matrix product," In SIGMOD, pp.107–122, 2015.

[28] D. Ting, "Approximate distinct counts for billions of datasets," In SIGMOD, pp.69–86, 2019.

[29] X. Wang, C. Qu, W. Wu, J. Wang, and Q. Zhou, "Are we ready for learned cardinality estimation?," PVLDB, vol.14, no.9, pp.1640–1654, 2021.

[30] Y. Wang, C. Xiao, J. Qin, X. Cao, Y. Sun, W. Wang, and M. Onizuka, "Monotonic cardinality estimation of similarity selection: A deep learning approach," In SIGMOD, pp.1197–1212, 2020.

[31] P. Wu and G. Cong, "A unified deep model of learning from both data and queries for cardinality estimation," In SIGMOD, pp.2009–2022, 2021.

[32] X. Wu, M. Charikar, and V. Natchu, "Local density estimation in high dimensions," In ICML, pp.5296–5305, 2018.

[33] X. Yan, J. Li, X. Dai, H. Chen, and J. Cheng, "Norm-ranging lsh for maximum inner product search," NIPS, pp.2952–2961, 2018.

[34] H.F. Yu, C.J. Hsieh, Q. Lei, and I.S. Dhillon, "A greedy approach for budgeted maximum inner product search," In NIPS, pp.5459–5468, 2017.

[35] B. Zheng, Z. Xi, L. Weng, N.Q.V. Hung, H. Liu, and C.S. Jensen, "Pm-lsh: A fast and accurate lsh framework for high-dimensional approximate nn search," PVLDB, vol.13, no.5, pp.643–655, 2020.

[36] Z. Zhou, S. Tan, Z. Xu, and P. Li, "Möbius transformation for fast inner product search on graph," In NeurIPS, pp.8218–8229, 2019.