公平性を考慮した高速な内積探索

青山　和禎† 　天方　大地† 　藤田　澄男†† 　原　隆浩†

† 大阪大学　〒 565–0871 大阪府吹田市山田丘 1-5
†† ヤフー株式会社
E-mail: †{aoyama.kazuyoshi,amagata.daichi,hara}@ist.osaka-u.ac.jp, ††sufujita@yahoo-corp.jp

あらまし　近年，推薦システムは多くの実世界アプリケーションに応用されている．既存の推薦システムでは，ユーザがアイテムに対して行う評価を予測し，評価予測値の高い上位 k 個のアイテムを探索する k-MIPS 問題を解くことが一般的である．しかし，k-MIPS 問題では公平性を考慮しておらず，推薦リストに含まれるアイテムに偏りがあり，推薦リスト中のアイテムの多様性が低く，新たな気付きを与えづらい．そのため，本論文では，公平性を考慮した内積探索問題を扱う．この問題は，カテゴリをもつアイテムベクトルの集合，クエリ，閾値および出力サイズ k が与えられたときに，k 個の各アイテムについて，カテゴリはランダムに決定され，決定されたカテゴリのアイテム集合からクエリとの内積が閾値以上となる全てのアイテムについて結果として出力される確率が均等になるように探索するというものである．素朴なアルゴリズムでは高速に解くことは難しいため，我々は，この問題を高速に解くためのアルゴリズムを提案する．我々のアルゴリズムは，コーシーシュワルツの不等式を用いたアイテムの探索範囲決めを二分探索を用いて高速に行い，決定した範囲内のアイテムからランダムに条件を満たすアイテムを探索する．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　内積探索，公平性

1　はじめに

近年，推薦システムは Amazon や Netflix など多くの実世界アプリケーションに応用されている．実世界においてユーザがアイテムの購入を検討する際，アイテム数は膨大であるため，ユーザが一つ一つ興味のあるアイテムを調べるのは手間である．そのため，ユーザが興味を持ちそうなアイテムだけを推薦する推薦システムを用いることでユーザがアイテムを探す負担を軽減することができる [4].

推薦システムにおける最も有名な手法の一つは MF（Matrix Factorization）であり，ユーザの未評価アイテムに対して評価予測値を得る手法である [9]．MF に基づく推薦システムでは，クエリとしてユーザベクトル $\mathbf{q}$ が与えられたとき，$\mathbf{q}$ との内積が最大となる上位 k 個のアイテムベクトルを探索する k-MIPS 問題を解くことが一般的である．この k 個のアイテムというのは，推薦リストとして実際に推薦されるアイテムである．

しかし，推薦結果に対するユーザ満足度の観点において，k-MIPS 問題を解くことが最も適しているとは限らない．まず，k-MIPS 問題では，推薦リストに含まれるアイテムには似たものが多くなり，ユーザの満足度が下がってしまう問題がある [7]．例えば，映画における推薦システムにおいて,『アベンジャーズ』に対して高い評価をつけたユーザに対して,『アベンジャーズ/エンドゲーム』など『アベンジャーズ』シリーズばかり推薦することは，ユーザにとって自明な推薦であり，新鮮さのない「飽き」を生じさせる推薦である．これは，ユーザの嗜好との合致度の観点から見ると，精度の良い推薦であるといえるが，ユーザ満足度の観点からみると，良い推薦であるとは言い難い．次に，k-MIPS 問題では，推薦結果が毎回同一の結果になってしまうという問題もある．推薦結果が毎回同一であるとユーザが新しい興味を得られる機会が少なくなってしまいユーザ満足度が低くなってしまう．実世界には，多種多様なアイテムが存在しており，推薦の幅が狭くなってしまうことはユーザにとって大きな機会損失である．

この問題を解決するために，我々は公平性を考慮した内積探索問題を考える．我々は 2 つの公平性を考慮することでユーザ満足度の高い推薦が行えると考える．1 つ目に，カテゴリの公平性を考える．k-MIPS 問題では，内積値の大きさに基づく推薦を行っており，特定のカテゴリのアイテムが多く推薦される可能性がある [7]．この問題を解決するため，推薦される k 個のアイテムの各カテゴリは推薦時に毎回均等な確率で決定されるという条件を設定する．この設定によって，推薦リスト内のカテゴリは多種多様なものとなり，推薦結果の多様性を高めることができる．2 つ目に，カテゴリ別のアイテム集合内の公平性である．k-MIPS 問題では，内積値の大きい順に k 個のアイテムを推薦しているが，それでは，推薦されうるアイテムの範囲が狭く，ユーザが推薦結果に新規性や多様性を感じない．そのため，我々はユーザとの内積がある閾値以上となる全てのアイテムについて，一様ランダムに推薦され，推薦結果は以前の推薦結果に依存しないという設定をする．つまり，ユーザがある程度興味のあるアイテムからランダムに推薦され，推薦結果は毎回ランダムなものとする．この設定により，推薦リストの

アイテムはユーザの嗜好を反映したものでありながら，ユーザにとって新規性や多様性の高いものにすることができる．

この問題を解く素朴なアルゴリズムとしては，クエリが与えられたとき，カテゴリをランダムに決定し，そのカテゴリのアイテム集合について，内積がある閾値以上となるアイテムを全て探索し，その探索したアイテム集合の中からランダムにアイテムを選択する．しかし，このアルゴリズムでは，k 個のアイテムを探索する場合，カテゴリ内の全てのアイテムについて，内積を計算する必要があり，カテゴリ内のアイテム数を n，ベクトルの次元数を d とすると $O(nd)$ の時間計算量が必要である．実世界のデータセットにおいて，アイテム数は膨大になりうるため，この素朴なアルゴリズムは大規模なシステムにはスケールしない．

以上から素朴なアルゴリズムでは，高速に計算することは難しい．そこで我々は，問題解決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，コーシーシュワルツの不等式と二分探索を用いて，条件を満たすアイテムの範囲を高速に求め，その中からランダムに探索していくというものである．我々の主な貢献は以下である．

- 内積空間における公平性を考慮した探索を行う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．具体的には，素朴なアルゴリズムでは時間計算量 $O(nd)$ だが，提案アルゴリズムでは $O(kd\log n)$ に削減できることを示す．
- 実世界のデータセットを用いて大規模な実験を行う．比較手法よりも高速であることを示す．

2 予 備 知 識

本章では，表記の説明と問題定義を行い，素朴なアルゴリズムでは高速に解くことが難しいことを説明する．

$\mathbf{P}$ は n 個のアイテムベクトル $\mathbf{p}_{(1)},...,\mathbf{p}_{(n)}$ からなるアイテム集合とする．$\mathbf{Q}$ は m 個のユーザベクトル $\mathbf{q}_{(1)},...,\mathbf{q}_{(m)}$ からなるユーザ集合とする．各データベクトル $\mathbf{p}\in\mathbf{P}$, $\mathbf{q}\in\mathbf{Q}$ は d 次元であり，$\mathbf{p}=(p_1,...,p_d)$, $\mathbf{q}=(q_1,...,q_d)$ である．カテゴリ数を C とする．各アイテムベクトルはカテゴリを 1 つ持ち，カテゴリ i のアイテムベクトル集合を $\mathbf{P}_i\in\mathbf{P}$ とする．

定義 1 （**内積**）2 つのベクトル $\mathbf{p}=(p_1,...,p_d)\in\mathbb{R}^d$ と $\mathbf{q}=(q_1,...,q_d)\in\mathbb{R}^d$ が与えられた時，内積 $\mathbf{p}\cdot\mathbf{q}$ は以下で求まる．

$$\mathbf{p}\cdot\mathbf{q}=\sum_{s=1}^{d}p_s\cdot q_s$$

.

閾値 τ が与えられたとき，内積 $\mathbf{p}\cdot\mathbf{q}\geqq\tau$ となるカテゴリ i のアイテム集合を $\mathbf{P}_i{}'$ とする．

ここで，我々の解決する問題を次のように定義する．

定義 2 （**公平性を考慮した内積探索問題**）クエリ $\mathbf{q}$，出力サイズ k，アイテム集合 $\mathbf{P}$ および閾値 τ が与えられた時に，$\mathbf{p}\cdot\mathbf{q}\geqq\tau$ となるアイテムの中から，次の条件で出力サイズ k 個のアイテムを探索する．

- k 個の各アイテムについて，あるカテゴリ i が探索される確率は均等である．
- 集合 $\mathbf{P}_i{}'$ において，集合内の各アイテムが探索される確率は均等である．
- クエリ結果は過去のクエリ結果とは独立である．

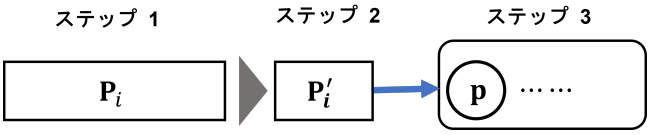

図 1: 素朴なアルゴリズム概要

この問題を解く素朴なアルゴリズムとしては，次のようなものが考えられる．図 1 は素朴なアルゴリズムの概要を示している．以下の処理を k 個のアイテムが得られるまで繰り返す:

(1) カテゴリをランダム一様に決定する．(決定されたカテゴリを i とする．)

(2) $\mathbf{P}_i{}'$ が未計算であれば，カテゴリ i のアイテム集合 $\mathbf{P}_i$ の全アイテムベクトルについて内積を計算し，閾値 τ 以上かどうかを調べ，$\mathbf{P}_i{}'$ を構築する．

(3) $\mathbf{P}_i{}'\neq\emptyset$ のとき，$\mathbf{P}_i{}'$ からランダムにアイテムベクトルを返し，そのアイテムベクトルを取り除く．

この素朴なアルゴリズムでは，全てのアイテムについて内積を計算する必要があるため，時間計算量 $O(nd)$ となってしまう．そのため，我々は時間計算量を削減し，$O(kd\log n)$ で解けるようなアルゴリズムを提案する．

3 提案アルゴリズム

この章では，公平性を考慮した内積探索問題を $O(kd\log n)$ 時間で解くことのできる我々の提案アルゴリズムについて説明する．まず，提案アルゴリズムの方針について説明し，次に，アルゴリズムをオフラインフェーズとオンラインフェーズに分けて具体的に説明する．最後に，アルゴリズムの分析として，空間計算量と，時間計算量および公平性について説明する．

3.1 方　　針

まず，素朴なアルゴリズムの時間計算量が $O(n)$ となってしまう原因は，カテゴリが決定された後，決定されたカテゴリのアイテム集合内の全てのアイテムベクトルについて，内積を計算するためである．我々の定義する公平性を考慮した内積探索問題では，$\mathbf{p}\cdot\mathbf{q}\geqq\tau$ となる全てアイテムベクトル $\mathbf{p}$ を求める必要はない．そのため，ランダムサンプリングを考える．具体的には，カテゴリをランダムに決定し（i とする），$\mathbf{P}_i$ からランダムサンプリングし，$\mathbf{p}\cdot\mathbf{q}\geqq\tau$ を満たす場合はそのアイテムベクトル $\mathbf{p}$ を返す．この処理を k 個のアイテムが得られるまで

繰り返す．ランダムサンプリングすることで，公平性の条件を満たしつつ，内積計算回数を減らすことができる．

しかし，$\mathbf{P}_i$ からランダムサンプリングした $\mathbf{p}$ が，$\mathbf{p}\cdot\mathbf{q} \geqq \tau$ となる確率は，$|\mathbf{P}_i'|/|\mathbf{P}_i|$ であるため，$|\mathbf{P}_i'|$ が $|\mathbf{P}_i|$ と比べて小さいとき，アイテムが返されるために必要なサンプリング回数が増え計算時間が長くなる．

そのため，我々は，コーシーシュワルツの不等式 ($\mathbf{p}\cdot\mathbf{q} \leqq \|\mathbf{p}\|\|\mathbf{q}\|$) を利用し，サンプリングした $\mathbf{p}$ が $\mathbf{p}\cdot\mathbf{q} \geqq \tau$ を満たす確率を高めることを考える．具体的には，コーシーシュワルツの不等式 ($\mathbf{p}\cdot\mathbf{q} \leqq \|\mathbf{p}\|\|\mathbf{q}\|$) から $|\mathbf{p}||\mathbf{q}| < \tau$ となる $\mathbf{p}$ を枝刈りする．条件を満たすことがないアイテムベクトルを枝刈りすることができるため，アイテムが返される確率が高まり，サンプリング回数を減らすことができるため，高速化できる．

次に，どのようにすれば高速に $\|\mathbf{p}\|\|\mathbf{q}\| < \tau$ となる $\mathbf{p}$ を枝刈りしサンプリング範囲を求めることができるのかということを考える．単純な方法として次のような手法が考えられる．

まず，前処理として，各カテゴリのアイテムベクトル集合において，アイテムベクトル $\mathbf{p}$ をノルム $\|\mathbf{p}\|$ の降順にソートしておく．次に，クエリフェーズとして，クエリ $\mathbf{q}$ が与えられたとき，カテゴリをランダムに決定し (i とする)，$\mathbf{P}_i$ 内のアイテムベクトル $\mathbf{p}$ について，順に $\|\mathbf{p}\|\|\mathbf{q}\|$ と τ とを比較していく．$\|\mathbf{p}\|\|\mathbf{q}\| < \tau$ となるアイテムベクトルが見つかったとき，そのアイテムベクトルと残りのアイテムベクトルは全て $\|\mathbf{p}\|\|\mathbf{q}\| < \tau$ となるため枝刈りを行うことができる．しかし，この手法では，最悪 $\mathbf{P}_i$ の全ての要素にアクセスする必要があり，サンプリング範囲を決定することに最悪 $O(n)$ 時間要する．これでは，高速化のための手段として十分ではない．

コーシーシュワルツの不等式を適用しサンプリング範囲を求めるということは，$\|\mathbf{p}\|\|\mathbf{q}\| \geqq \tau$ を満たすアイテムベクトル $\mathbf{p}$ の内，$\|\mathbf{p}\|$ が最小のものを求めるということである．そのため，二分探索が適用でき，$O(\log n)$ 時間で求めることができる．

我々のアルゴリズムは，コーシーシュワルツの不等式と二分探索を用いて，条件を満たすアイテムの範囲を高速に求め，その中からランダムに探索していくことで，効率的に条件を満たすアイテムベクトルにアクセスする．

3.2 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 に適用できる．

オフラインアルゴリズムとしては，アイテム集合 $\mathbf{P}$ を各カテゴリのアイテム集合 $\mathbf{P}_1 \cdots \mathbf{P}_C$ に分割する．各カテゴリのアイテム集合 $\mathbf{P}_1 \cdots \mathbf{P}_C$ 内の全てのアイテムベクトル $\mathbf{p}$ について，それぞれノルム $\|\mathbf{p}\|$ を計算する．各カテゴリのアイテムベクトル集合をノルム $\|\mathbf{p}\|$ でソートする．

3.3 クエリフェーズ

次に，クエリとして $\mathbf{q}$ が与えられたとき，オフラインフェーズで構築したデータ構造を用いて，高速に内積探索するためのアルゴリズムについて説明する．図 2 はクエリフェーズの概要を示している.Algorithm 1 はクエリフェーズの擬似コードを示している．

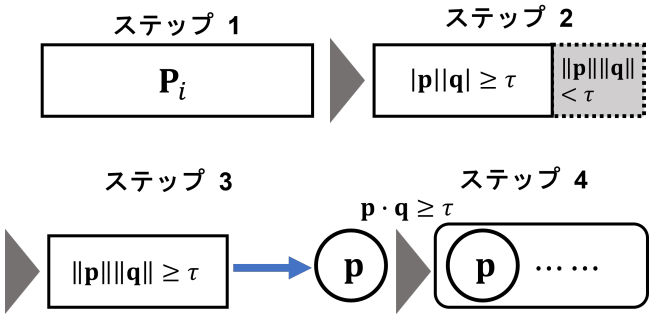

図 2: クエリフェーズ概要

Algorithm 1 クエリフェーズ

Input: $\mathbf{P}_1, \ldots, \mathbf{P}_C$, $\mathbf{q}$, τ, and k
Output: $\mathbf{Q}_k$

```
compute ||q||
Q_k ← ∅
while |Q_k| < k do
    i ← a randomly selected category
    index ← arg min_{l, p_l ∈ P_i} ||p_l|| ||q|| ≧ τ
    for each j ∈ [1, O(log n)] do
        p_α ← random sample from {p_1, ··· , p_index}
        if p_α · q ≧ τ then
            Q_k ← Q_k ∪ {p_α}
            break
        end if
    end for
end while
```

クエリフェーズでは，以下の処理を k 個のアイテムが所得されるまで繰り返す．

(1) ランダムにカテゴリを決定する．(i とする．)

(2) カテゴリ i のアイテムベクトル集合 $\mathbf{P}_i$ について，コーシーシュワルツの不等式を適用し，二分探索で $\|\mathbf{p}\|\|\mathbf{q}\| < \tau$ となる $\mathbf{p}$ について枝刈りし，探索範囲を決定する．

(3) (2) で求めた探索範囲の中から $\mathbf{p}$ をランダムにサンプリングする．

(4) (3) で取得した $\mathbf{p}$ が $\mathbf{p}\cdot\mathbf{q} \geqq \tau$ となり，結果の集合に含まれていなければ，結果の集合に $\mathbf{p}$ を追加する．そうでなければ，(3) に戻る．サンプリングは最大 $O(\log n)$ 回行う．

3.4 分　　析

空間計算量．我々のアルゴリズムにおいて，アイテムベクトル集合 $\mathbf{P}$ 内の n 個のアイテムベクトルとノルムをデータ構造として用いている．そのため，空間計算量は $O(n)$ である．

時間計算量．j をクエリフェーズ (2) で二分探索によりサンプリング範囲を求めたときの $\|\mathbf{p}\|$ が最小のアイテムベクトルのイ

ンデックスとする．

定理 1　我々の提案アルゴリズムの時間計算量の期待値は $O(k(\log n + d))$ である．

証明.　クエリフェーズの *(2)* の処理は二分探索を用いることで $O(\log n)$ 時間で実行できる．次にクエリフェーズにおいて，$\mathbf{p} \cdot \mathbf{q} \geqq \tau$ を満たす $\mathbf{p}$ が k 個返されるために必要なサンプリング試行回数について考える．既に k' 個 $\mathbf{p}$ が取得されているときを考える．ここで $A_{k'}$ を $\mathbf{p} \cdot \mathbf{q} \geqq \tau$ を満たす $\mathbf{p}$ が返されるために必要な試行回数とする．$A_{k'}$ は変数である．ここで，$|Q|$ を決定したカテゴリ内の $\mathbf{p} \cdot \mathbf{q} \geqq \tau$ を満たす $\mathbf{p}$ の個数とする．

$$\Pr(A_{k'} = m) = (\frac{j - |Q| + k'}{j})^{m-1} \times \frac{|Q| - k'}{j}$$

ここで，$z = 1 - \frac{|Q|-k'}{j}$ とすると，

$$E[A_{k'}] = \sum_{m=1}^{\infty} m z^{m-1}(1-z) = \frac{1}{1-z}$$
$$= \frac{j}{|Q| - k'} = O(1)$$

ここで，$j = O(n)$，$|Q| = O(n)$，$k' = O(k)$ である．よって，k 個のアイテムが返されるための試行回数の期待値は，$\sum_{k'=0}^{k-1} E[A_{k'}] = O(k)$．よって，時間計算量の期待値は $O(k(\log n + d))$ となる．　□

定理 2　少なくても定数確率で提案アルゴリズムの時間計算量は $O(k(\log n + d))$ となる．

証明.　クエリフェーズの *(3)* の処理について考える．既に k' 個 $\mathbf{p}$ が取得されているときを考える．ここで B をクエリフェーズ *(3)* においてランダムに $\mathbf{p}$ を決定したとき，$\mathbf{p} \cdot \mathbf{q} < \tau$ である事象とする．ここで $\Pr(B) = 1 - \frac{|Q|-k'}{j}$ である．s 回連続で失敗する確率は，ユニオンバウンドより，

$$\Pr[B \cup \ldots \cup B] \leqq \sum_{s} \Pr[B] = s(1 - \frac{|Q| - k'}{j})$$

ここで，$s(1 - \frac{|Q|-k'}{j}) = c$ とする．$c \in (0, 1]$ であり，$c = O(1)$ とする．このとき，$s = O(1)$ を得る．そのため、時間計算量は $O(k(\log n + d))$ となる．　□

定理 3　少なくても $(1 - \frac{1}{n})^k$ の確率で提案アルゴリズムの時間計算量は $O(kd\log n)$ となる．

証明.　クエリフェーズの *(3)* の処理について考える．既に k' 個 $\mathbf{p}$ が取得されているときを考える．ここで s 回試行したとき，少なくとも 1 回は成功する確率は，$1 - (1 - \frac{|Q|-k'}{j})^s$ である．これが $1 - \frac{1}{n}$ と等しいとき，$s = \frac{\log n}{\log \frac{j}{j - |Q| + k'}} = O(\log n)$ となる．よって k 個のアイテムを取得するとき，$(1 - \frac{1}{n})^k$ の確率で時間計算量 $O(kd\log n)$ となる．　□

公平性．まず，カテゴリの公平性について考える．カテゴリの変数を X とする．i と j を任意のカテゴリとすると $\Pr(X = i) = \Pr(X = j)$ となり，カテゴリの公平性の条件を満たす．また，$\mathbf{p}$ と $\mathbf{p}'$ を集合 $\mathbf{P}_i{}'$ 内の任意のアイテムベクトルとする．このとき，アイテムベクトルの変数を Y とすると $\Pr(Y = \mathbf{p}) = \Pr(Y = \mathbf{p}')$ となるため，カテゴリ別のアイテム集合内の公平性の条件を満たす．

4 評 価 実 験

この章では，評価実験を行い，我々のアルゴリズムは公平性を考慮した内積探索問題を解く他のアルゴリズムに比べて，高速であることを示す．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．

4.1 設　　定

データセット.　我々は以下の 4 つの実世界データセットを用いた．各データセットのユーザ数，アイテム数，およびカテゴリ内アイテム数を表 1 に示す．

- Netflix[1]: Netflix Prize で用いられたレーティングのデータセット．
- Movielens[2]: Movielens 25M データセット．
- AmazonKindle [6]: AmazonKindle の書籍のレーティングのデータセット．
- AmazonMovie [6]: AmazonMovie の映画のレーティングのデータセット．

ユーザベクトルとアイテムベクトルを生成するために，[3] の Matrix Factorization を実行した．各データセットのデータベクトルの次元数は 200 である．Movielens のデータセットはカテゴリをメタ情報を基に割り当てている．その他の 3 つのデータセットは，データセット内の各アイテムにランダムにカテゴリを割り当てている．Movielens のカテゴリ数は 20 であり，その他の 3 つのデータセットのカテゴリ数は 25 である．

比較手法.　我々は以下の 3 つの比較手法を用いた．

- Cauchy-Schwarz: 提案アルゴリズムから二分探索とランダムサンプリングを取り除いたアルゴリズム．時間計算量は $O(n)$.
- r-NNS [2]: 公平性を考慮した範囲検索アルゴリズム．時間計算量の期待値は，$o(kdn\log n)$.
- r-NNIS [1]: クエリ独立性をもたせた公平性を考慮した範囲検索アルゴリズム．時間計算量の期待値は，$o(kdn\log^5 n)$.

すべてのアルゴリズムは C++で実装を行い，g++ 7.5.0 に最適化オプション-O3 付加してコンパイルした．

評価指標として，$\mathbf{p} \cdot \mathbf{q} \geqq \tau$ となる $\mathbf{p}$ が 1 つ以上存在する全ユーザの平均実行時間 (msec) を用いた．

1：https://www.cs.uic.edu/liub/Netflix-KDD-Cup-2007.html
2：https://grouplens.org/datasets/movielens/

表 1: データセットの統計

	ユーザ数	アイテム数	カテゴリ内アイテム数
Netflix	480,189	17,770	710 ∼ 711
Movielens	162,541	59,047	51 ∼ 15191
AmazonKindle	1,406,890	430,530	17221 ∼ 17222
AmazonMovie	2,088,620	200,941	8037 ∼ 8038

4.2 結果と分析

提案アルゴリズムと比較手法の比較．表 2 に $\tau = 4$，$k = 5$ での平均実行時間の結果を示す．提案アルゴリズムの方が，最大 (AmazonKindle の場合) 約 333 倍，最低 (Netflix の場合) 約 18 倍高速であることが分かる．Cauchy-Schwarz と比較して提案アルゴリズムが高速である理由は，全てのアイテムベクトルを探索することなく，条件を満たすアイテムベクトルを効率的に探索できるためである．提案アルゴリズム，r-NNS，および r-NNIS はサンプリングする手法であるが，r-NNS と r-NNIS はサンプリングプールから条件を満たすアイテムベクトルが取得される確率が低く，多くのアイテムベクトルについて探索を行う必要があり，提案アルゴリズムと比較して低速となる．

表 2: $\tau = 4$，$k = 5$ での平均実行時間 (msec) の結果

	Proposed	Cauchy-Schwarz	r-NNS	r-NNIS
Netflix	2.33×10^{-2}	0.422	0.570	1.69
Movielens	3.51×10^{-2}	1.76	2.66	11.6
AmazonKindle	7.68×10^{-2}	25.6	5.20	24.8
AmazonMovie	4.18×10^{-2}	12.3	3.12	12.5

τ の影響．図 3 に τ を 3，3.5，4 および 4.5 に変化させたときの提案アルゴリズムと比較手法の平均実行時間の結果を示す．どのデータセットにおいても，提案アルゴリズムが最も高速であるということが分かる．また，τ を大きくするほど Cauchy-Schwarz は高速になっていることが分かる．これは，τ が大きくなるほど，多くのアイテムベクトルが $\mathbf{p} \cdot \mathbf{q} < \tau$ となるため，多くのアイテムベクトルを枝刈りでき，高速に計算できるためである．提案手法，r-NNS，および r-NNIS は．τ を大きくするほど，$\mathbf{p} \cdot \mathbf{q} \geqq \tau$ となる $\mathbf{p}$ が少なくなるため低速となっている．また，同じアイテムをサンプリングすることも多くなるため，τ が大きくなるほど低速となる．

k の影響．図 4 に k を 5，10，15，20 に変化させたときの提案アルゴリズムと比較手法の平均実行時間の結果を示す．どのデータセットにおいても，提案アルゴリズムが最も高速であるということが分かる．どの手法においても，k を大きくするほど，多くのカテゴリのアイテム集合について探索を行うため，低速となる．

5 関 連 研 究

5.1 k-MIPS

k-MIPS 問題を解くことは，多くのアプリケーションにとって重要であるので，高速に解くために多くの研究がなされている．ここでは，厳密解ベースの手法と近似解ベースの手法の二つに分けて説明する．

厳密解ベースの手法には，空間分割を行う手法と，線形スキャンによる手法の 2 つがある．空間分割を行う手法として，[12] では木データ構造に基づく分枝限定法を用いて不必要なデータベクトルを枝刈りする Dual-Tree および Dual-Tree の効率を上げるための新しいデータ構造 Cone Tree が提案されている．しかし，空間分割に基づく手法は次元の呪いによって高次元空間においてはうまく機能せず，数十次元まで次元が増加すると，線形スキャンよりも性能が悪くなる．そのため，線形スキャンベースの手法が多数提案されている．線形スキャンベースの手法として，FEXIPRO [10] や LEMP [13, 14] がある．FREXIPRO は SVD 変換，正数変換，および整数近似によって厳密な内積上限を設定し，枝刈りの効果を高める手法である．LEMP は，アイテムベクトルの集合をバケットに似たようなノルムのベクトルが含まれるようにバケットに分割し，バケット毎にノルムの大きさに応じた最適な手法を選択することで高速化している．

高次元ベクトル空間の場合，k-MIPS 問題を厳密に解くことは計算コストが高いため，近似解ベースの手法も多く研究されている．一般的な手法として，内積探索問題を最近傍探索問題や最大コサイン類似度探索問題へと変換し，LSH を用いて高速に解く手法がある．Simple-LSH [11] は，アイテムとクエリそれぞれに対称変換を行い，MIPS 問題を MCS 問題 (最大コサイン類似度問題) に変換する．H2-ALSH [8] は，前処理フェーズにおいて，ノルムの大きさに応じてデータベクトルの集合をいくつかの集合に分割しハッシュテーブルに格納する．クエリフェーズにおいて分割したサブセットについて，ノルムの降順でサブセット内のデータベクトルの個数に応じて，線形スキャンと LSH を用いた近似最近傍探索の内適切な手法を用いている．また，k-MIPS 問題から近似最近傍探索問題へと変換するためのクエリとデータベクトルの変換に歪み誤差の少ない QNF 変換を用いることでより厳密解に近づくようにしている．また，[5] ではベクトル量子化を用いた手法 ScaNN を提案している．ベクトル量子化は，データベクトル集合を有限個のセントロイドに置き換えることで，大量のデータを少ない代表パターンに置き換え圧縮する手法である．ScaNN は，このベクトル量子化によって生じる損失を最小限にしつつ，ベクトルの量子化を行う．

しかし，これらの問題は k-MIPS 問題を解く手法であり，我々の定義する公平性を考慮した内積探索問題を解くことはできない．

5.2 公 平 探 索

r-NNS [2] は，クエリ $\mathbf{q}$ とデータ集合 $\mathbf{D}$ が与えられたとき，$\mathbf{q}$ との距離が r 以内の各データ点 $\mathbf{p} \in \mathbf{D}$ について，一様な確率で返されるようなデータ構造を構築する問題に取り組んだ手法である．r-NNS のアイデアは，データ集合 $\mathbf{D}$ 内のデータ $\mathbf{p}$ について，ランダムなランク付けを行い，LSH を用いて高速化する手法である．時間計算量は，$o(kn \log n)$ となる．r-NNIS [1] は，

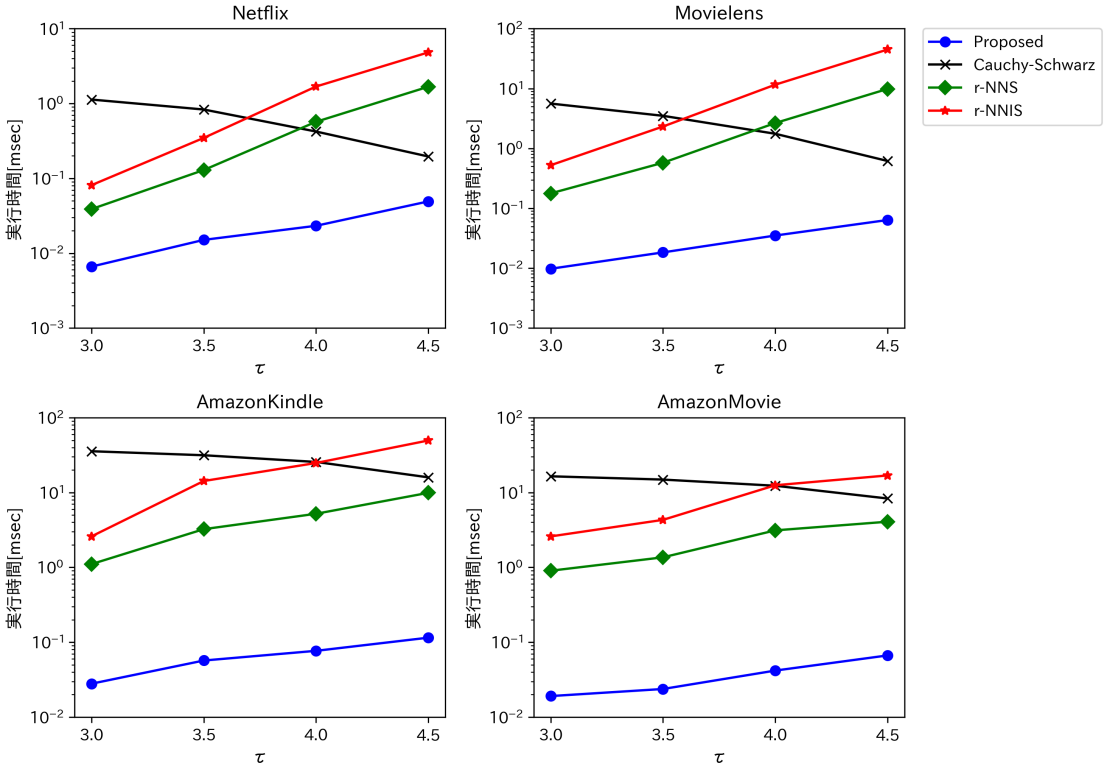

図 3: τ の影響

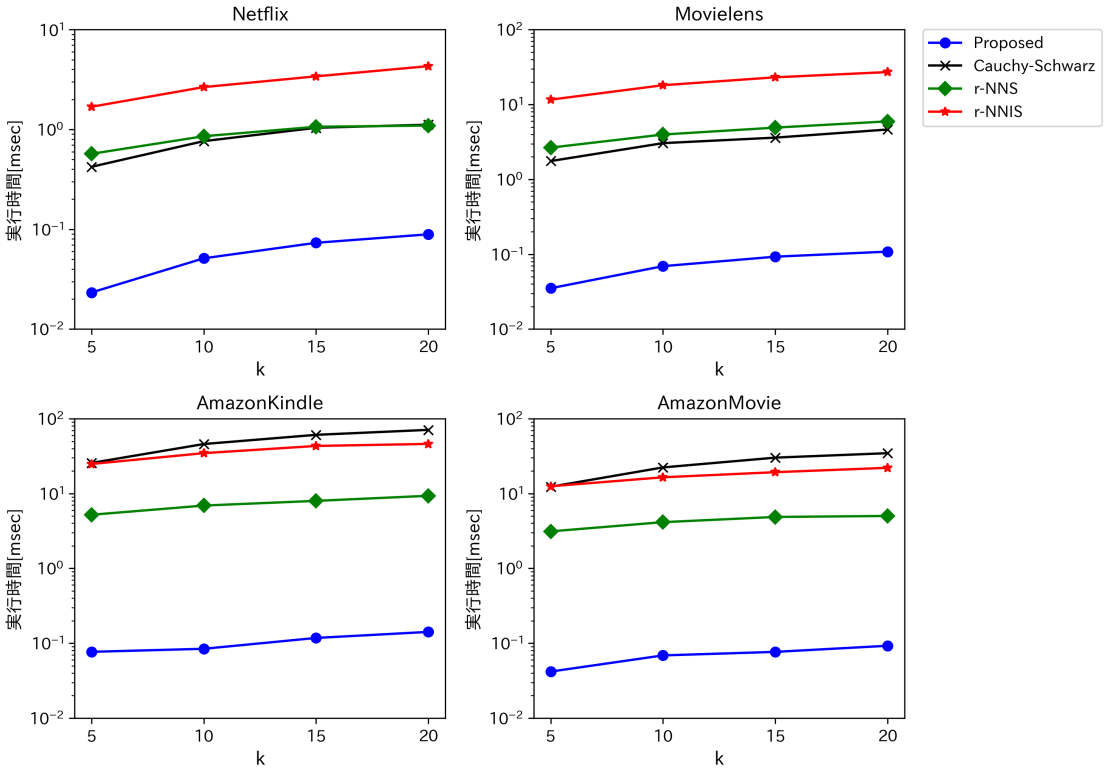

図 4: k の影響

r-NNS の問題にクエリ結果が過去のクエリ結果とは独立であるという条件を追加した問題に取り組んだ手法である．r-NNIS のアイデアは，データ点をランダムシャッフルし，いくつかのセグメントに分割する．クエリが与えられたとき，ランダムにセグメントを選び探索を行う．これを LSH を用いて高速化している．r-NNIS の時間計算量は，$o(kdn\log^5 n)$ となる．

これらの手法は，LSH を用いた手法であるため，常に条件を満たすデータ点 $\mathbf{p}$ は一様な確率でアクセスされることを保証し

ていない．

6 結　　論

本論文では，公平性を考慮した内積探索問題を定義した．この問題では，カテゴリの公平性とカテゴリ内のアイテムの公平性を考慮することで，推薦リスト内のアイテムの多様性や新規性を高めることができる．素朴なアルゴリズムでは，$O(n)$ 時間かかる．我々はコーシーシュワルツの不等式を用いたアイテムの探索範囲決めを二分探索を用いて高速に行い，決定した範囲内のアイテムからランダムに条件を満たすアイテムを探索することで，$O(kd \log n)$ 時間で解くことができるアルゴリズムを提案した．また，実世界データセットを用いた実験を行い，我々のアルゴリズムが比較手法を大幅に上回る性能を持つことを確認した．

謝　　辞.

本研究の一部は，文部科学省科学研究費補助金・基 盤研究 (A)(18H04095)，JST さきがけ (JPMJPR1931)，および JST CREST(JPMJCR21F2) の支援を受けたものである．

文　　献

[1] M. Aumuller, S. Har-Peled, S. Mahabadi, R. Pagh, and F. Silvestri, “Fair near neighbor search via sampling,” In SIGMOD-Record, pp.42–49, 2021.
[2] M. Aumüller, R. Pagh, and F. Silvestri, “Fair near neighbor search: Independent range sampling in high dimensions,” In PODS, pp.191–204, 2020.
[3] 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.
[4] P. Cremonesi, Y. Koren, and R. Turrin, “Performance of recommender algorithms on top-n recommendation tasks,” In RecSys, pp.39–46, 2010.
[5] 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.
[6] 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.
[7] K. Hirata, D. Amagata, S. Fujita, and T. Hara, “Solving diversity-aware maximum inner product search efficiently and effectively,” In RecSys, pp.198–207, 2022.
[8] 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.
[9] Y. Koren, R. Bell, and C. Volinsky, “Matrix factorization techniques for recommender systems,” in IEEE Computer, pp.30–37, 2009.
[10] 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.
[11] B. Neyshabur and N. Srebro, “On symmetric and asymmetric lshs for inner product search,” In ICML, pp.1926–1934, 2015.
[12] P. Ram and A.G. Gray, “Maximum inner-product search using cone trees,” In SIGKDD, pp.931–939, 2012.
[13] 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.
[14] C. Teflioudi, R. Gemulla, and O. Mykytiuk, “Lemp: Fast retrieval of large entries in a matrix product,” In SIGMOD, pp.107–122, 2015.