属性改良問題の高速化に対する検討

此島 魁二† 陳 漢雄† 天笠 俊之† 古瀬 一隆††

† 筑波大学 〒 305–8571 茨城県つくば市天王台 1-1-1

†† 白鷗大学 〒323-8585 栃木県小山市駅東通り 2 丁目 2-2

E-mail: † s2020590@u.tsukuba.ac.jp, {chx, amagasa}@cs.tsukuba.ac.jp, †† furuse@fc.hakuoh.ac.jp

あらまし 近年、情報検索の技術を用いて評価者が好みそうな商品をランキング形式で求めるという研究や、商品の開発者がどのような商品が好まれるかということを検索する研究が注目を集めている。この問題に対する先行研究では、商品を他の評価者と比べて相対的に高く評価する評価者を対象とし、評価者のランキングにおける対象商品のランクの上昇量によって判定を行う手法が提案された。しかしこちらの手法は商品を相対的に高く評価する評価者を求める検索を複数回呼び出すため、計算の粒度を細かくした場合、非常に実行時間がかかるといった問題点が生じている。本研究では既存研究の問題点を解決すべく、ランク上昇量による評価を行いながらも、高速に改良候補を計算する手法の提案を行う。具体的には、評価者のランキングをすべて計算し、それらを保持しておくことで再計算にかかる時間を削減する手法である。この手法について貪欲な手法、先行研究との比較を行い、精度と実行時間の面での有用性の検証を行う。

キーワード 情報検索、情報推薦、Top-k Query、Improved Query

1. はじめに

近年、商品の開発者が商品の評価データを用い、自分たちの商品を競合商品がある中でより多くの評価者に好まれるように改良するという試みがなされてきている。これにはレビューサイトなどで評価者からの商品の評価データが数多く得られるようになってきていることが関係している。

このような問題を考えるために改善対象商品の属性値の改良を提案する属性改良問題という研究が考えられてきた。これは評価者の好みと商品の評価集合のデータを与え、改善対象商品をクエリとして与えたときに、より多くの評価者に好まれるような改良を提案するというものである。

属性改良問題を解くアルゴリズムとして既存研究である Reverse Rank Improved Query[1][2]が挙げられる。この手法は既存手法である Improved Query[3]の発展である。IQ では Top-k Query[4]による評価を行っていたが、RRIQ ではランク上昇量に着目し、評価を行うことで IQ の問題点であった改良として認識されないケースが存在するという点を解決した。

しかし、RRIQ の問題点として網羅的に改良戦略を作成し、それらに対して何度も Reverse k-Rank Query(RkR)[5][6][7]を用いてランクを求めるため、特に高次元において計算時間が増大するという点が挙げられる。

この問題を解決するために本研究では RRIQ の高速化に取り組む。先行研究で問題であった RkR の呼び出しによる実行時間の増加を抑えるために既存研究である Iterative RkR Extracting をベースとし、各評価者について前もって商品のランキングを作成する。改良戦略の評価を判定する際にそのランキングを用い、より多くの評価者でランクが上昇しているものを有望な改良戦略として求める。

2. 問題及び諸定義

先行研究である IQ(2.2 節で詳述)では特に Top-k query を利用して条件の設定を行ってきた。Top-k query は一人の評価者の好みを与えたときに、その評価者がもっとも好みそうな k 個の商品を求めるような検索である。評価者が好みと商品の評価をベクトルで表し、その内積を計算することでランクを求める。ランクの定義を以下に示す。

定義 1(rank(u,q)) 商品データ集合を P、評価者の好みデータを u、改善対象商品 q を与えたとき

$rank(u,q) = |S|$.

ただし、 $S \subseteq P, \forall p_i \in S, \forall p_j \in (P-S): f(u,p_i) \leq f(u,q) \wedge f(u,q) < f(u,p_j)$

である。

RkR は相対的に商品を高く評価する評価者 k 人を絞り込む検索手法である。これを用いて上位 k 人においてのみ厳密なランク計算を行い、計算の削減を行う。以下に RkR の定義を示す。

定義 2 (Reverse k-Rank Query) 商品データ集合を P、評価者の好みのデータ集合を U、正の整数 k、改善対象商品 q を与えたときに *Reverse k-Rank Query* は次

のデータ集合 S を求める。$S \subseteq U, |S| = k, \forall u_i \in S: \forall u_j \in (U - S): rank(u_i, q) \leq rank(u_j, q)$ が成り立つ。

商品データ集合を P、評価者の好みのデータ集合を U、各商品 $p_i \in P$が d 次元のベクトルで d 個の属性値を持ち、$p_i = (p_i^{(0)}, p_i^{(1)}, \dots, p_i^{(d)})$というベクトルで表されるとする。また、各評価者 $u_j \in U$が d 個の好みを持つ d 次元ベクトル $u_j = (u_j^{(0)}, u_j^{(1)}, \dots, u_j^{(d)})$であり、$\sum_{k=0}^{d} u_j^{(k)} = 1$であるとする。改良戦略の定義を定義 3、RRIQ の定義を定義 4 に示す。

定義 3(改良戦略) 商品ベクトルに対する改良戦略を s とする。改良戦略は d 次元のベクトルとして $s = (s^{(0)}, s^{(1)}, \dots, s^{(d)})$と表す。改良戦略 s をもとの商品ベクトル p_iに適用した改良後商品ベクトル p_i'を $p_i' = p_i + s$で表す。

定義 4 (Reverse Rank Improved Query)
商品ベクトルに対する改良戦略候補を Sとする。このとき RRIQ では以下の改良戦略ベクトル s_jを求める。

$$s_j = argmax_{s_i \in S}\{eval(q + s_i)\}$$

3. 関連研究

3.1. Reverse k-Rank Query

Reverse k-Rank Query は Top-k query の逆側からの検索の Reverse Top-k query[8]の発展である。Reverse Top-k query はクエリとして商品を与えたときに全評価者と商品について内積を計算し、クエリ商品を上位 k 番以内に評価する評価者集合を返すといった検索方法であるが、クエリ商品の需要が小さい場合、上位 k 番目以内に評価する評価者が存在しないといった場合が起こりうる。発展である Reverse k-Rank query は相対的に高く評価する評価者を返すため、必ず上位 k 人の評価者を取得できる。

3.2. Improved Query

ここでは、Top-k query を例にクエリ検索がどのように行われるかの例を示し、先行研究の IQ が Top-k query を利用してどのように改良戦略を提案するのかを示す。表 1 のような商品の属性値に対する評価の集合、表 2 のような評価者の好みベクトルの集合があるとする。このとき、ベクトルの内積値、k=2 としたときの Top-k query は表 3 のようになる。先行研究に倣って、ベクトルの値、内積値ともに小さい方が高い評価としている。

表 1 商品ベクトル集合

	cpu	メモリ	容量
p1	0.5	0.3	0.4
p2	0.3	0.3	0.4
p3	0.2	0.7	0.1

表 2 評価者の好みベクトル集合

	cpu	メモリ	容量
u1	0.3	0.3	0.4
u2	0.1	0.2	0.7
u3	0.5	0.4	0.1

表 3 Top-2 query

	u1	u2	u3	u1内のランク	u2内のランク	u3内のランク
p1	0.4	0.39	0.41	3	3	3
p2	0.34	0.37	0.31	2	2	1
p3	0.31	0.23	0.39	1	1	2

このとき、u1 の Top-2 query は{p2,p3}、u2 の Top-2 query は{p2,p3}、u3 の Top-2 query は{p2,p3}となる。ここで改良戦略として $s = \{-0.1, -0.1, -0.1\}$を p1 に適用する。適用後の Top-2 query は表 4 から、u1 は{p1,p3}、u2 は{p2,p3}、u3 は{p1,p2}となる。これより p1 の Top-k query ヒット数が改良前 0 に対して、改良後は 3 に増加している。よってこの改良は効果的であると判断できる。先行研究では予算内で Top-k query のヒット数を最大限増加させる調整(Max-Hit IQ)と、与えられた Top-k query のヒット数を満たすように最小限のコストで改良を行う調整(Min-Cost IQ)の 2 通りの方法が提案された。

表 4 改良戦略適用後の Top-2 query

	u1	u2	u3	u1内のランク	u2内のランク	u3内のランク
p1	0.3	0.29	0.31	1	2	1
p2	0.34	0.37	0.31	3	3	1
p3	0.31	0.23	0.39	2	1	3

3.3. Reverse Rank Improved Query

IQ では 2 通りの改良の調整が提案されていたが、そのどちらも問題点を抱えている。1 つ目に、予算が少ない場合に、最小限の Top-k query ヒット数増加を満たすような改良戦略が存在せず、改良戦略が求められないという問題がある。またもう一つの問題として IQ では改善対象商品の Top-k query ヒット数で改良戦略の評価を行うため、すでに k 番目以内に入っている商品が順位を上昇させた場合や、商品の順位が上昇したが、k 番目以内にはならなかった場合には改良されたと認識されない。その問題点を解決するために提案された

ものが Reverse Rank Improved Query(RRIQ)である。RRIQ では商品の評価をランク上昇量により行う。これにより、前述の問題点を解消することができる。先行研究では RRIQ 問題の解法として Iterative RkR Extracting(IRkRE)法([6]の 3.3 参照)が提案されている。

IRKRE では、改善対象商品を改良するための予算ベクトルを作成し、それらの中から評価値がより高いものを抽出する。そして再度予算を分配し、改良戦略を決定する。このときに用いられる予算分配アルゴリズムを Algorithm 1 に示す。このアルゴリズムでは予算を一定間隔で分割したものを各属性に振り分け、予算ベクトル候補を作成する。この操作を粒度を細かくしながら繰り返し行うことで、精度の良い近似解を得ることができる。

Algorithm 1 Create Improved Vector

Input: b, d, δ, $Record = (0, 0, ..., 0)$
Output: *a set of vector with all possible combination within budget b*

```
initialize current_dimention ← d only when starting the function
for element = 0; (budget − element) > 0; element+ = δ) do
    Record[current_dimention − 1] ← element
    sum ← accumlate dimention values up to (current_dimention − 1)
    tmp = b − sum
    if current_dimention == 2 && tmp >= 0 then
        record[0] = tmp
        result ← record
    else if current_dimention == 2&&tmp < 0 then
        current_dimention = d
        break
    else
        recursively call self function using (record, current_dimention − 1, δ, b)
    end if
end for
```

4. 提案手法

4.1. 提案手法の概要

本研究は RRIQ 問題について先行研究である IRkRE 手法を発展させ、RkR によるランク再計算回数を削減することで高速化を図る。また、IQ、の問題点であった上位のランク上昇が改良効果として判定されないという点、IRkRE の問題であった上位におけるランク上昇と下位におけるランク上昇を等価に扱うという点を解決するために新たな評価式を導入する。そして、この評価式に適したアルゴリズムを提案する。提案アルゴリズムでは、ランク再計算回数を抑えるため、各評価者における商品のランキングを取得し、保持する操作を行う。その後、改良戦略を作成し、保持したランキングを用いて改良戦略が有望であるかの判定を行い、有望なものは候補として残す。これを繰り返し行い、改良戦略を決定する。

4.2. 評価式

本研究では商品ベクトルと評価者の好みベクトルは各属性値が$0 \leq \ p_i^{(m)} \leq 1$, $0 \leq \ u_j^{(n)} \leq 1$ $(0 \leq m, n \leq d)$を満たすものであるとし、計算の都合上 0 に近いほど良い評価とする。また、評価関数は(1)の式で与えられる。

$$eval(q+s) = \sum_{k=0}^{|U|-1} \frac{rank(u_k, q) - rank(u_k, q+s)}{rank(u_k, q+s)} \quad \ldots (1)$$

この評価式は上昇したランクが大きく、また上位のランク上昇であればあるほど、高い値をとるような調整がなされている。

4.3. アルゴリズム

提案する手法は大きく分けて 4 つのステップに分けられる。概要を図 1 に示す。Step 1 では評価者全てに対して商品ランキングを作成し。保持する操作を行う。Step 2 では Iterative RkR Extracting 法で用いられる予算分配法(Algorithm 1)を用いて網羅的に近似改良戦略を作成する。Step 3 では作成した改良戦略に対して内積を計算し、各評価者のランキングと比較を行う。これによって改良後の商品の正確なランクが得られる。これを全評価者に対して計算することでその改良戦略の評価を得る。そしてその中でより評価が高いものを有望な改良として残す。この Step2 から Step3 を一定の精度が得られるまで繰り返し、最後に RkR を用いて厳密な解を求める。

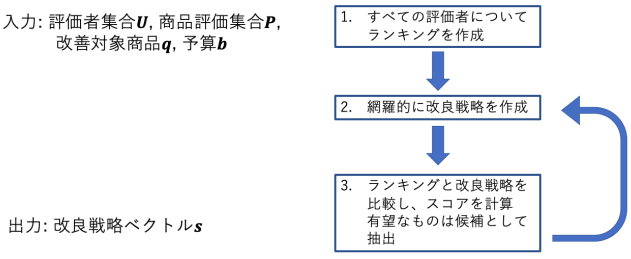

図 1 提案手法の概要

Step 1 では R-Tree 上における Top-k query 検索[9]を利用し、特定の k の値までの順位のランキングを各評価者に対して作成する。ここで、ランクと商品だけでなく、評価者ベクトルとその商品ベクトルの内積値をまとめて保持する。これによって後述する Step 3 で

二分探索による検索が可能になる。

Step 2 では一定間隔で分割した予算を配分し、網羅的に近似改良戦略を作成することを考える。この Step では IRkRE 法の改良戦略候補作成と同様に改良戦略候補を作成する。

Step 3 では Step 1 で作成した商品のランキングと Step 2 で作成した改良戦略候補を用いてスコアの比較を行う。まず、評価者の Top-k Query に入っているかどうかを判定し、Top-k Query に入っていた場合には正確なランク上昇量を求める。ランク上昇量の検索をには、Step 1 で保持した内積値を用いて、二分探索を行う。ランクの中央値と改良後の商品の内積値を比較し、

$$(\text{改良後の商品}) > (\text{ランクの中央値})$$

であれば、ランクの上位半分を残し、そうでなければ下位半分を残すという操作をランキングの総数が 1 になるまで行い、最後に残ったランクと改良後の商品を比較してランクを決定する。これによって効率よく、また正確に改良後の商品のランクを求めることができる。これを各評価者に対して行った後に、改良後のランクと評価式を用いて正確な評価値を算出し、全ての評価者に対して評価値の和を計算する。その中からよりよい評価値を得られた改良戦略を有望な改良として残す。これを予算の分割間隔を小さくしながら繰り返し行うことで最適な改良戦略を求める。これらの操作を疑似コードにまとめたものを Algorithm 2 に示す。

Algorithm 2 Improved Query

Input: $P, U, q, b, d, \delta, k, times$
Output: Improved query

```
currentBudget ← b
currentDelta ← δ
lastBudgetVec ← (0, 0, ..., 0)
for i = 0; i < |U|; i++ do
    topkresults ← topkquery(|P|, u[i], k)
end for
for j = 0; j < times; j++ do
    if i ≠ times then
        b ← b − currentDelta * d
    end if
    budgetVec ← createImprovedVector((0, 0, ..., 0), d,
    currentDelta, currentBudget))
    budgetVec ← all combinations sum of
    budgetVec and lastBudgetVec
    for l = 0; l < budgetVec.size(); l++ do
        for m = 0; m < |U|; m++ do
            caliculate score budgetVec[l] for u[m]
            if score in Top − k Query then
                caliculate ranking by binary search
            end if
        end for
        caliculate sums[l]
    end for
    lastBudgetVec ← extract Top − x budgetVec
    with sums
end for
caliculate result with lastBudgetVec
return result
```

5. 実験、考察

提案手法の有用性を示すために、貪欲に改良戦略を作成し適用する方法、IRkREと提案手法(Top-k Query による絞り込みを用いたものとそうでないもの)について実験を行い、その結果を示した。実験環境は以下の通りである。

- OS: MacOS Monterey 12.1
- CPU: 2.3GHz Intel Core i5
- メモリ: 8GB 2133 MHz LPDDR3
- 開発言語: C++

5.1. 実験データ

実験は商品ベクトル集合のデータとして実データ、一様分布の合成データを用い、評価者の好み集合のデータとして一様分布の合成データを用いた。実データには Airbnb のホテルデータセット[10]を用いており、(*満足度,収容人数,価格*)の 3 次元を抽出し、各属性に対して正規化の処理を行った。実験データのベクトルの各属性は 0 から 1 の値をとるように調整されており、0 に近いほど高い評価であるとしている。また、実験設定は表 5 の通りである。

表 5 実験データ設定

パラメータ	既定値	範囲
次元数d	3	
評価者集合$\|U\|$	10000	10000-50000
商品集合$\|P\|$	1000	1000-5000
予算b	1	
分割間隔δ	0.1	
ループ回数$times$	2	
ループ時の抽出件数	$\log_d \|budgetVec\|$	
取得商品数k	100	

5.2. 改良効果関数

予算がどの程度評価に反映されるかを決定するために、本研究では入力を予算ベクトル、出力を商品の改良ベクトルとした改良効果関数を定める。合成データにおける改良効果関数は各属性とも予算の半分が改良ベクトルとして反映されるものとした。また、実デー

タにおける改良効果関数は表 6 の通りである。

表 6 改良効果関数

属性値	改良効果関数
満足度	$log_{11}(1+2*b_1)$
収容可能人数	$b_2/2$
価格	$b_3/5$

5.3. 実行時間

提案手法(k=100, |P|)、IRkRE(ITERATIVE)手法、貪欲な手法(GREED)の 3 手法について評価者、商品数を変化させながら実験を行った。図 2, 3 が合成データによる実験結果、図 4,5 が実データによる実験結果である。

実験結果より、提案手法は評価者数の増加に対して他手法よりも高速に処理を行えていることがわかった。これは他手法が RkR を繰り返し呼び出すのに対して、提案手法でははじめに Top-k Query を保持しておくことで呼び出し回数を削減できているからだと考えられる。

また、k=100 としたときの提案手法は商品数の増加に対しても実行時間を抑えられており、IRkRE よりも良いパフォーマンスを発揮している。これは Top-k Query による絞り込みのおかげで、商品数の増減に関わらず、計算量が一定になるためである。

図 2 の結果より、提案手法はどちらも評価者数の増加に対してほぼ線形に実行時間が推移しており、ある程度大規模なデータセットに対しても実行時間の増加を抑えられると考えられる。実世界におけるシステムの運用を考えた場合、

$$(\textit{評価者数}) \gg (\textit{競合商品数})$$

であるため、提案手法は他 2 手法に比べて有用性が高いと考えられる。しかし、図 3 より商品数が増加した場合には IRkRE 手法が k=|P|のときの提案手法よりも実行時間の増加が抑えられている。

この原因として、提案手法ではランキングをすべて求めるため、商品数の増加に伴い実行時間が増加していくが、IRkRE 手法では商品集合におけるランキングではなく、改良候補内におけるランキングを用いて候補の評価を行うため、繰り返しで呼び出される RkR の計算量は商品数の増加に関わらず常に一定で済む。このことから商品数の増加に対しては IRkRE 手法のほうが k=|P|のときの提案手法よりも良い結果になったと考えられる。

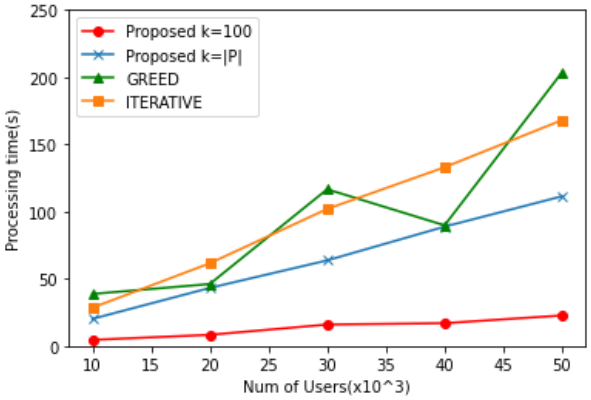

図 2 合成データにおける|U|の変化

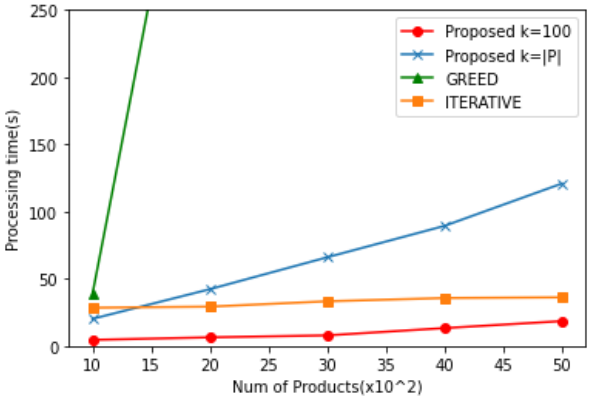

図 3 合成データにおける|P|の変化

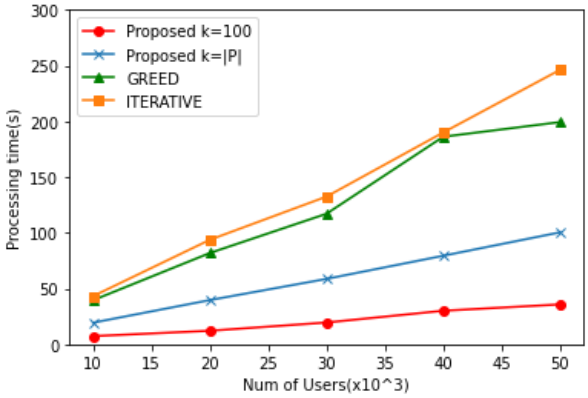

図 4 実データにおける|U|の変化

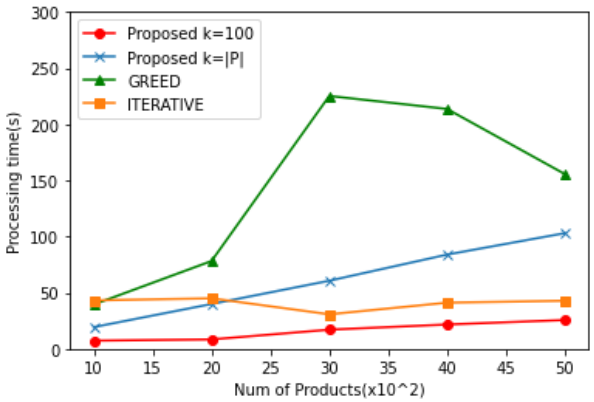

図 5 実データにおける|P|の変化

5.4. 精度

提案手法が他手法と比べてどの程度精度を保てているかを検証するために、各手法で同一の商品を改良する際の評価式の値の一人あたりの平均を用いて比較検証を行った。グラフは貪欲な手法に対しての割合を示したものであり、その結果を図 6~9 に示す。

表 7, 10 より評価者数が変化したときの提案手法の評価値は貪欲な手法に近い値をとっており、実データ、合成データともに評価者が増加したとしても高い精度で近似解を得られることを示している。また、提案手法において k の値をある程度小さい値にしたとしても、精度を保てている。

しかし、表 8, 10 より商品数を変化させたときの評価値は貪欲な手法の評価値に大きく及ばない場合がいくつか見られる。特に、$k = 100$ のとき、精度が落ちるという問題点が生じている。

これについては大きく 2 つの原因が考えられる。1 つ目に、Top-k Query による絞り込みの結果、k 番目以下までのランク上昇を無視しているという点である。この問題は特に改良商品の元のランクが低く、また改良効果が小さいときに起こりやすいと考えられる。改良商品のランクが低い場合、ランクを上昇させたとしても多くの評価者にとって k 番目以内に好む商品にならない場合が想定される。また、改良効果関数や予算の影響で改良効果が小さいときにも、ランクの上昇が小さくなるため、評価者が k 番目以内に好む商品とならない可能性が高い。そのような場合、本研究のアルゴリズムでは最適解を求められない可能性がある。

2 つ目にループ時の抽出件数の影響が考えられる。本研究と先行研究の IRkRE ではループを行い、有望な改良戦略を作成する予算ベクトルを抽出する。その過程で最適解とは全く異なった傾向のベクトルを抽出している場合、解の精度が悪くなる。このようなことが起こる原因としてループの途中における改良戦略の評価値が同じ値となっており、抽出がうまくいかなかったのではないかと考えられる。

今後の展開として、k の値を変えながら実験を行い、最適な値を発見することや、抽出件数の調整を行い、実行時間をなるべく大きくしない範囲で精度を高めるなどの検証を行う必要がある。

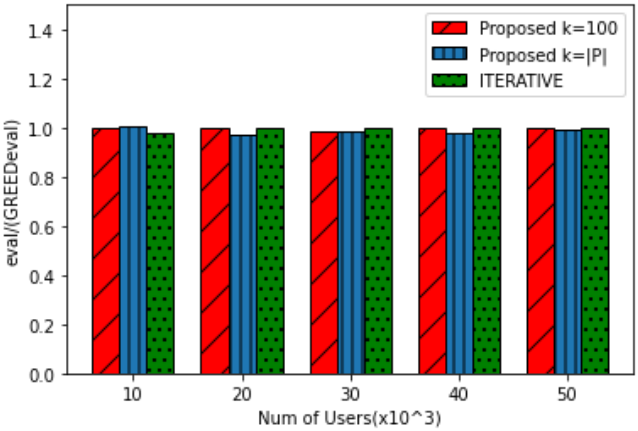

図 6 評価者変化、合成データにおける評価値

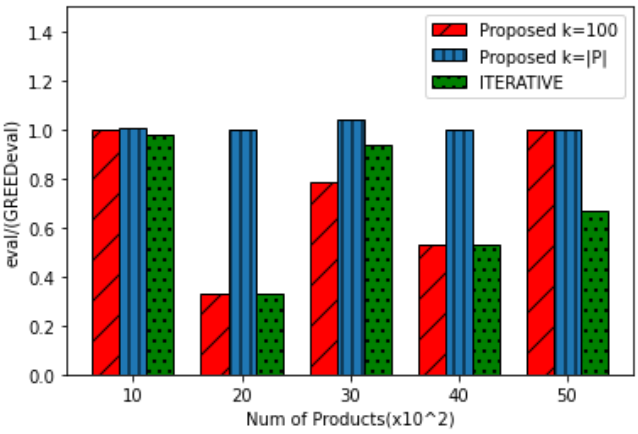

図 7 商品数変化、合成データにおける評価値

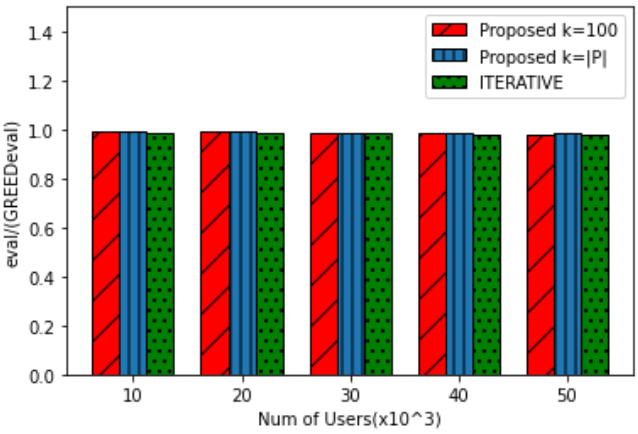

図 8 評価者数変化、実データにおける評価値

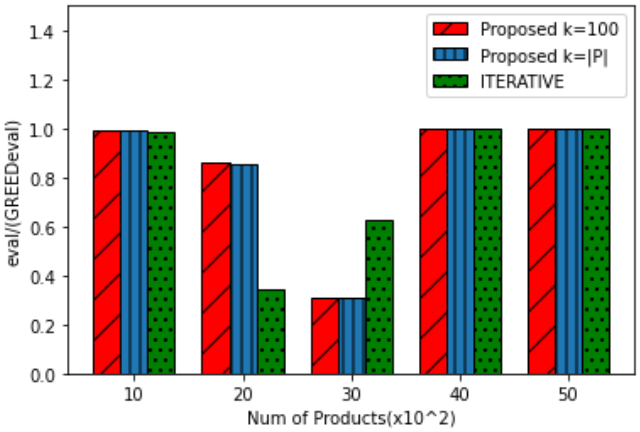

図 9 商品数変化、実データにおける評価値

6. おわりに

本研究では、先行研究で問題であった大規模データにおける実行時間の増加を改善するために、評価者のランキングを用いて属性改良問題を効率的に解く手法を提案した。今後の改善としては商品数、評価者の増加による実行時間の変化に対応するために、評価者集合をクラスタリングし、クラスタごとに評価者を抽出してそれらのランキングから近似的な解を求めるといったものが挙げられる。また、現在はデータのインデックスに R-tree を用いているが、他のインデックス手法についても検討する必要があると考えられる。

謝辞

本研究は JSPS 科研費 19K12114 の助成を受けたものである。

参 考 文 献

[1] 小栗 大輝, 董 于洋, 陳 漢雄, 古瀬 一隆. 改良コスト最小化の逆情報推薦. DEIM Forum 2018 D3-5.

[2] 山下 雅弘,佐野 ひかり, 陳 漢雄, 古瀬 一隆, 北川 博之. 逆ランク検索による情報改良支援. DEIM Forum 2018 D4-3.

[3] Guolei Yang, Ying Cai. Querying Improvement Strategies. EDBT. pp.294–305, 2017.

[4] Ihab F. Ilyas, George Beskales, and Mohamed A. Soliman. A survey of top-k query processing techniques in relational database systems. ACM Comput. Surv. 40, 4, Article 11, pp.1-58. 2008.

[5] Zhao Zhang, Cheqing Jin, and Qiangqiang Kang. 2014. Reverse k-ranks query. Proc. VLDB Endow. 7, 10, pp.785–796. 2014.

[6] Yuyang Dong. Hanxiong Chen, Kazutaka Furuse, Hiroyuki Kitagawa. Aggregate Reverse Rank Queries. DEXA 2016, pp. 87-101. 2016.

[7] Yuyang Dong. Hanxiong Chen, Kazutaka Furuse, Hiroyuki Kitagawa. Grid-Index algorithm for reverse rank queries. EDBT 2017, pp. 306-317. 2017.

[8] Vlachou, A., Doulkeridis, C., Norvag, K., Kotidis, Y.: Branch-and-Bound Algorithm for Reverse Top-k Queries. SIGMOD 13, pp. 481–492. 2013.

[9] Yufei Taoa, Vagelis Hristidisb, Dimitris Papadiasc, Yannis Papakonstantinou.:Branch-and-bound processing of ranked queries. ScienceDirect Volume 32, 3, pp. 424-445. 2007.

[10] Tom Slee (2022-02-10) Airbnb Data Collection: Get the Data. http://tomslee.net/airbnb-data-collection-get-the-data (閲覧日: 2022-02-10)