DEIM Forum 2018 D3-5

改良コスト最小化の逆情報推薦

小栗　大輝† 　董　 于洋†† 　陳　 漢雄†† 　古瀬　一隆††

† 筑波大学 情報学群 情報メディア創成学類　〒 305–8550 茨城県つくば市春日 1-2
†† 筑波大学 システム情報系　〒 305–8571 茨城県つくば市天王台 1-1-1
E-mail: †s1411443@u.tsukuba.ac.jp, ††touuyou@gmail.com, ††{chx,furuse}@cs.tsukuba.ac.jp

あらまし　情報推薦はユーザの意思決定を効率よく支援する。今までの研究は例えばユーザが好みそうな商品をランキングの形式で求めるという方法で顧客に商品・情報を薦める。一方、本研究の目標は逆に商品の販売者側からの情報検索の応用であり、情報供給者・生産者に品質向上に有益な情報を提供する。具体的に、より多くのユーザに好まれるようなクエリ改良 (Improvement query, IQ) を提案する。また、検索がより高速に出来るよう索引手法を利用した高速化手法についても検証を行う。

キーワード　逆情報推薦，情報検索，インデックス

1. はじめに

クエリによる情報検索技術の一つとして、一つのオブジェクトをクエリとして与え、対象のユーザ群により好まれるような属性値の改良を提案する Improvement query(以下、IQ) [1] という研究が近年なされている。これは、これまで活発に研究が行われてきたクエリによる情報検索技術である、Top-k query [2] や Reverse k-rank query(以下、RkR) [3] [4] [5] のさらなる応用である。クエリによる情報検索は、マーケティングにおいて有効なアプローチであり Top-k query は一人のユーザの好みをクエリとして与えた時に、そのユーザがもっとも好みそうな k 個の商品を求めるような顧客側からの検索に使用される。逆に、RkR は商品を一つクエリとして与えた時に全商品と全ユーザに対してスコアを計算し、その結果から、クエリ商品を相対的に高く評価する k 人のユーザを求めるような検索手法であり、これは商品の販売者側が潜在的な顧客を検索する手法として利用できる。

先行研究で提案された IQ は、特に Top-k query の応用である。クエリとして商品を与えた時に、全ユーザについて Top-k query を計算し、クエリ商品の Top-k query へのヒット数を元にクエリ商品をある条件下で改良する改良戦略を求めるような検索手法である。本研究では、先行研究 IQ を応用し改良戦略の評価にランク上昇量を利用することで、既存手法の問題点を改善し、より検索利用者のニーズにあった逆情報推薦を実現することや、提案手法を索引手法を用いたフィルタリングにより高速化することを目的とする。

1.1 問題定義

商品のデータ集合を P、ユーザ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、商品 p_i の各属性値は $p_i = (p_i^{(1)}, p_i^{(2)}, \ldots, p_i^{(d)})$ とする。そして各ユーザ $u_j \in U$ が d 個の好みを持つ d 次元のベクトルであり、ユーザの好みは $u_j = (u_j^{(1)}, u_j^{(2)}, \ldots, u_j^{(d)})$ として、$\sum_{k=1}^{d} u_j^{(k)} = 1$ とする。本研究では、$q \in P$ をクエリ商品として一つ与えた時に、条件を満たす改良戦略を効率良く検索することを目的とする。改良戦略の定義を定義 1 に示す。

図 1　Improvement query の簡単なイメージ図

定義 1 (改良戦略)　オブジェクトに対する改良戦略を s とする。改良戦略 s は d 次元のベクトルとして $s = (s_1, s_2, \ldots, s_d)$ と表す。改良戦略 s を元のオブジェクト p_i に適用した改良後オブジェクト p_i' は、$p_i' = p_i + s$ で与えられる。

本研究は提案手法の高速化に Reverse k-rank query による計算対象のフィルタリングを利用する。定義 2 に Reverse k-rank query の定義を示す。

定義 2 (Reverse k-rank query)　商品のデータ集合 P、ユーザの好みデータの集合 U、正整数 k、クエリ商品 q を与えた場合。*Reverse k-rank query* は次のデータ集合 S を求める。即ち、$S \subseteqq U$, $|S| = k$ であり、$\forall\, u_i \in S : \forall\, u_j \in (U - S) : Arank(u_i, Q) \leqq Arank(u_j, Q)$ が成り立つ。

先行研究では IQ を求める手法として Min-Cost IQ と Max-Hit IQ の二つの方法が提案された。Min-Cost IQ は与えられた任意の数の Top-k query にヒットする改良戦略を最小コストで検索する手法である。Top-k query へのヒットとは、クエリ商品がユーザの評価上位 k 番目以内に含まれるということを表す。また、Max-Hit IQ はある予算が与えられた際にその予算内で Top-k query のヒット数が最大となるような改良戦略を検索する手法である。Min-Cost IQ と Max-Hit IQ の定義を定義 3、定義 4 でそれぞれ示す。定義 3、定義 4 の関数 $H_k(q)$ はクエリ商品

$q \in P$ の Top-k query のヒット数を表す。

定義 3 (Min-Cost IQ) オブジェクト改良の目標として *Top-k query* の最小ヒット数 $\tau \in \mathbb{I}$、クエリ商品 $q \in P$、クエリ q を改良した時にかかるコストを表すコスト関数 $Cost_q$ が与えられた時、$H_k(q+s) \geqq \tau$ かつ $Cost_q(s)$ が最小となるような q に対する改良戦略 s を求める。

定義 4 (Max-Hit IQ) 予算 $\beta \in \mathbb{R}$、クエリ商品 $q \in P$、クエリ q を改良した時にかかるコストを表すコスト関数 $Cost_q$ が与えられた時、$Cost_q(s) \leqq \beta$ かつ $H_k(q+s)$ が最大となるような q に対する改良戦略 s を求める。

1.2 モチベーション

IQ は前述したように Top-k query の性質を利用して改良戦略を求めることを前提としている。そのため、Top-k query が持つ特性による問題点の影響を受けてしまい、改良戦略検索の際に不具合が生じてしまうことがある。

まず、一つ目の問題点としてクエリ商品の需要が非常に小さいパターンが考えられる。このような商品がクエリとなると、商品が改良されているにもかかわらず、ユーザにとって上位 k 番目以内に好む商品となるまで改良するには、非常に多くの改良が必要となりコストが膨大となってしまう。また、クエリ商品によるもう一つの問題点として、あるクエリ商品がユーザの好み k 番目にわずかに届かない状態である場合も思うように改良の効果が得られない。例えば、図 2 に示すように改良戦略によってランクが変化したとする。この場合、先行研究の評価手法では 2 倍のヒット数となり、高い効果があると期待されるが、実際にユーザに対して改善ができているとは言い難く、不自然な結果となってしまう。

改良戦略適用前

u_1	9
u_2	11
u_3	12
u_4	12
u_5	13
u_6	11
u_7	11
u_8	9
u_9	8
u_{10}	12

Top-10 queryヒット数4
平均ランク10.8

改良戦略適用後

u_1	8
u_2	10
u_3	9
u_4	9
u_5	11
u_6	11
u_7	9
u_8	9
u_9	8
u_{10}	7

Top-10 queryヒット数8
平均ランク9.1

図 2　改良戦略適用前後のランク変化

そこで、このような改良戦略を求める際の評価値としてランク上昇量に着目することで、既存手法の IQ の問題点を解決し、柔軟な検索のニーズに答えられるような改良戦略検索手法を提案する。また、Reverse k-rank query の性質を利用した提案手法の高速化を行い、実験により実行速度の変化や改良の効果を示す。

2. 関 連 研 究

先行研究の IQ [1] では、Top-k query のヒット数を用いた改良戦略の求め方について定義がされている。また、改良後オブジェクトの Top-k query のヒット数を効率よく求める手法として Efficient Strategy Evaluation(ESE) という手法が提案された。以下では ESE を用いた Min-Cost IQ と Max-Hit IQ について詳しく述べる。

2.1 Efficient Strategy Evaluation(ESE)

ESE では、オブジェクトのヒットする Top-k query の個数を求めるために、各オブジェクトの境界線を関数として定義し、その関数の入力としてそれぞれの Top-k query を利用する。この境界線をもとにサブドメインを形成し、改良戦略 s 適用後のサブドメインの変化に含まれる Top-k query に対してのみスコアの再計算を行うことで、計算時間の短縮を実現する。図 3 は、2 次元の場合の改良戦略適用による境界線の移動、表 1 はそれによるオブジェクト間の順位変動を表したものである。

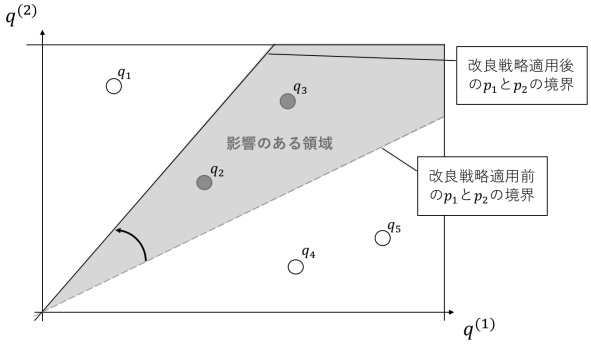


図 3　改良戦略の適用と境界線の変化

表 1　オブジェクト改良後のランク変化

クエリ	ランキング結果	
	s 適用前	s 適用後
q_1	f_2, f_1	f_2, f_1
q_2, q_3	f_1, f_2	f_2, f_1
q_4, q_5	f_1, f_2	f_1, f_2

2.2 Improvement 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 2〜表 4 では、Top-k query の $k = 2$ の例を示している。以下の表では、商品の評価の指標としてデザイン・スペック・ブランドの 3 つが属性値として設定されているとし、表 2 では、ユーザ $u_1 \sim u_3$ の商品の属性値に対する好みの比重を表している。デザイン、スペック、ブランドそれぞれに割り振り、各ユーザ毎に合計が 1 となる値とする。表 3 は 3 つの商品 $p_1 \sim p_3$ に対して、それぞれの属性値に対する評価を示している。表 4 は、全商品、全ユーザに対してベクトルの内積計算によって各スコアを計算したものであり、先行研究に則りスコアが小さいものを上位としてラン

ク付けを行った結果を示す。これらの結果を踏まえ、例として p_2 をクエリ商品とすると、Top-2 query にヒットするのは p_2 のスコアがランク 2 以上にある $\{u_1, u_3\}$ となる。

表 2　ユーザの好み

ユーザ	デザイン	スペック	ブランド
u_1	0.2	0.3	0.5
u_2	0.6	0.3	0.1
u_3	0.1	0.7	0.2

表 3　商品の評価

商品	デザイン	スペック	ブランド
p_1	2	3	4
p_2	3	3	3
p_3	4	1	2

表 4　ユーザと商品による Top-2 query

商品	u_1	u_2	u_3	u_1 内のランク	u_2 内のランク	u_3 内のランク
p_1	3.3	2.5	3.1	3	1	3
p_2	3.0	3.0	3.0	2	3	2
p_3	2.1	2.9	1.5	1	2	1

IQ は、オブジェクトに改良戦略を適用することで Top-k query のヒット数を増やし、その増加数を指標として最適な改良戦略を見つける。先ほどのクエリ商品である p_2 に改良戦略 $s = \{-1, -2, 0\}$ を適用することを考える。ここでも、属性値が小さい方がより高いスコアが出るという規則に則っているので、改良戦略はマイナスの値を持つこととしている。適用後改めて全商品、全ユーザに対して各スコアを計算した結果を表 5 に示す。

表 5　改良戦略適用後の Top-2 query

商品	u_1	u_2	u_3	u_1 内のランク	u_2 内のランク	u_3 内のランク
p_1	3.3	3.5	3.1	3	3	3
p_2'	2.2	1.8	1.5	2	1	1
p_3	2.1	2.9	1.5	1	2	1

表 5 の結果から、p_2' をクエリ商品として Top-2 query にヒットするものは $\{u_1, u_2, u_3\}$ となり、改良戦略適用により Top-2 query のヒット数が 2 から 3 へと増加している。よって改良戦略は効果的であると判断できる。先行研究では、このようなクエリ改良のアプローチとして、Top-k query に任意の値ヒットする改良戦略を最小コストで求める Min-Cost IQ と与えられた予算内で最大の Top-k query にヒットする改良を提案する Max-Hit IQ の 2 種類の方法が提案された。

2.3　Reverse k-rank query

Reverse k-rank query の前身となるクエリ検索として、Reverse Top-k query [8] がある。これは、クエリ商品を与えた時、全商品と全ユーザについてスコアを計算しクエリ商品を上位 k 位以内と評価するユーザを求めるような手法である。Reverse Top-k query の問題点として、クエリ商品の需要が少ない時に解の数が極端に少ない、もしくは全くの 0 となってしまうことが挙げられる。このような問題点を解決するために提案されたのが Reverse k-rank query [3] [4] [5] である。Reverse k-rank query は、クエリ商品を与えた時、全商品と全ユーザについてスコアを計算し、クエリ商品を相対的に高く評価するようなユーザ k 人を求めるようなクエリ検索である。本研究では、改善オブジェクトのランク上昇量に着目した IQ を提案する。そのために、対象オブジェクトの平均ランクを求める必要があるため、Reverse k-rank query の相対的な上位 k 人のユーザ集合に必ず絞り込めるという性質を利用してランキングの再計算を行うことで高速化を図る。

2.4　R-tree

RkR では、ユーザの好みデータ集合と商品集合からクエリ商品を相対的に高く評価するユーザを効率よく計算するために、内部で R-tree [6] と呼ばれる空間インデックス手法が使われている。R-tree は各節点が最大 M 個の異なる子節点へのリンクを保持する木構造であり、すべての情報は葉に存在し、そのそれぞれは最大 M 個の異なる n 次元空間オブジェクトを蓄えられる。R-tree の定数として、もう一つ $m \leqq M/2$ が定義され、R-tree は、$O(log_m(n))$ で挿入、削除、クエリをサポートする。本研究の解説として、2 次元の R-tree 利用例を提示するが、自然に n 次元へと拡張することができる。[7] R-tree は、階層的に入れ子となっている重なり合った最小外接矩形 (MBR) で空間を分割することで空間内のオブジェクトを効率的に検索できるようにする。例えば、2 次元かつ $M = 3$ の場合の R-tree 利用による空間分割を考える。図 4 は 2 次元空間に R-tree による MBR での空間分割を適用したものである。図 4 内の円は空間内のオブジェクトを表し、灰色、オレンジ、緑の長方形は分割された MBR を表す。R1〜R14 は MBR を一意に表す識別子となっている。図 4 のように空間を分割すると、R-tree は図 5 のような構造となる。R-tree の葉ノードは各オブジェクトへの参照を持っており、図 5 の黄色い円で示されるオブジェクトを検索する際には R_1 から順に対象オブジェクトが含まれる MBR を $R_1 \rightarrow R_4 \rightarrow R_9$ の順で絞り込みを行い、対象オブジェクトにアクセスする。

3.　提 案 手 法

本研究では、Top-k query に依存せずランク上昇量を指標として改良戦略を求める方法、そしてそのような改良戦略の求め方に対して Reverse k-rank query を用いた高速化手法を提案する。

3.1　コスト関数

本研究では、改良戦略を求める際にクエリ商品に改良戦略を適用した時のコストを考える。コストは先行研究と同様にクエリ検索時に利用者から与えられたコスト関数を元に計算をする。

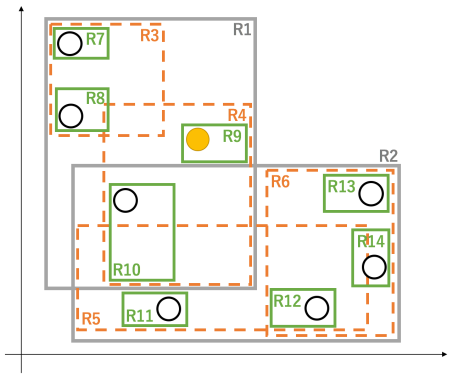


図 4　R-tree による 2 次元空間の分割例

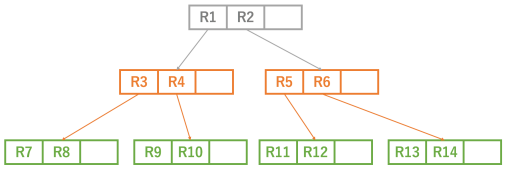


図 5　2 次元空間分割に伴うインデックスの作成例

関数としてコストをモデル化するのは一般的な考えであり [9]、本研究の実装には独自に用意したコスト関数を用いることとする。クエリ商品 q と改善戦略 s が与えられた時、ランダムな重みベクトル $w = (w_1, w_2, ..., w_d)$ を用いて本研究で使用するコスト関数を式 1 に示す。ここで、ランダムな重みベクトル w を用いるのは、例えばクエリ商品としてパソコンを考えた時に、CPU を強化するのとメモリを強化するのとでは、同じように商品が改良されたとしても、実際にかかるコストが大きく変わってくる。このような属性値毎にかかるコストの差を擬似的に表現するために重みベクトル w を用いる。本研究で使用するコスト関数は、属性値の元の値が高いほど改良に大きなコストがかかり、また適用する改良戦略が大きいほどコストが高くなるように設定がなされている。さらに、先行研究に則りクエリ商品の属性値は $[0, 1]$ の範囲に正規化されており、0 に近いほど評価が高いとするため、そのための処理を式 1 内では行っている。

$$Cost_q(s) = \sum_{j=1}^{d} \{(1 + (1 - q^{(j)})) * s_j * w_j\} \qquad (1)$$

3.2　Reverse Rank IQ

Reverse Rank IQ では、改良ベクトル $s = (s_1, s_2, ..., s_d)$ を用いて、$p_i' = p_i + s$ となる改良オブジェクトを作成する際のオブジェクト改良の指標として、改良ベクトル適用前後のクエリ商品の全ユーザに対しての平均ランクを用いる。

定義 5 (RR IQ)　ユーザの好みデータの集合 U、オブジェクト改良の目的として平均ランク上昇量 $\tau \in \mathbb{R}$、クエリ商品 q、クエリ商品 q を改良した時にかかるコストを表すコスト関数 $Cost_q$、商品 q のあるユーザにおいてのランクを求める関数 r_q が与えられた時、

$$\frac{\sum_{j=1}^{|U|} \{(r_q(u_j) - r_{q'}(u_j))/r_q(u_j)\}}{|U|} \geqq \tau \qquad (2)$$

を満たし、$Cost_q(s)$ が最小となるような s を求める。

3.3　RR IQ アルゴリズム

上記定義 5 を踏まえた上で、提案手法 IQ をどのように解いていくのかを説明する。IQ を求める問題は NP 困難であり [1]、NP 困難な問題の最適解を求めるのには大変な時間がかかる。そのため本研究では、最適解ではなくとも十分にその近似解を求められるようなヒューリスティックなアルゴリズムを提案する。

Step 1. あるユーザ u_i とクエリオブジェクト q を用いて改善戦略の作成を行うとする。まず、q の属性値 $\{q^{(1)}, q^{(2)}, \ldots, q^{(d)}\}$ の中から一つを選びその属性に任意の一定の値を与え、それ以外は 0 である改善戦略候補をコスト関数に適用することを考える。これによって、対象オブジェクトの属性値の中で最も改良コストの小さい属性を見つける。

Step 2. 対象オブジェクトよりランクがひとつ上のオブジェクトとのスコアの差を計算し、このスコアの差の分、先ほどの最低改良コストの属性値を改良した改良戦略を作成する。これを全ユーザに対して繰り返し行い、改良戦略候補を作成する。

Step 3. この改良戦略候補からコストがもっとも低いものを最適な改良戦略として q へ適用する。この一連の操作を任意のランク上昇量 τ を満たすまで続ける。

以上 Step1〜Step3 を疑似コードとしたものを Algorithm 1 に示す。7–10 行目で対象オブジェクトの中から最小コストの属性値を見つける処理を行う (Step1)。11–13 行目でランクがひとつ上のオブジェクトとのスコアの計算と改良戦略候補の作成を行い、6–14 行の for 分でそれを全ユーザに対して繰り返す (Step2)。4–18 行目でランク上昇量が τ を超えるまで対象オブジェクトへの改良戦略の適用を行う (Step3)。

表 6　RR IQ アルゴリズム内で使用する記号とその意味

記号	意味
P	商品集合
U	ユーザの好みデータ集合
q	クエリ商品
τ	目標平均ランク上昇量
s	改良戦略
RkR	RkR によるユーザデータのフィルタリングをする関数
$Arank$	クエリ商品の平均ランクを求める関数
S	改良戦略の候補の集合
$score_u$	クエリ商品のあるユーザに対してのスコアを求める関数

Algorithm 1 Reverse Rank IQ

Require: P, U, q, τ
Ensure: Improvement strategy s

$q' \leftarrow q$
$U' \leftarrow$ Result of $RkR(P, U, q)$
/* 以下 RkR IQ は U の代わりに U' を用いる */
while $Arank(q') - Arank(q) < \tau$ **do**
  $S \leftarrow \emptyset$
  for $u_j \in U'$ **do**
    for attribute $q^{(k)} \in P$ **do**
      sample strategy s_k
      Find s_k with minimal $Cost_q(s_k)$
    end for
    Compute score q and one rank above $p \in P$
    Create s with $score_{u_j}(q + s) > score_{u_j}(p)$
    S.add(s)
  end for
  Find $s \in S$ with minimal $Cost_q(s)$
  $q' \leftarrow q' + s$
  Compute $Arank(q')$
end while
Return $s = q' - q$

3.4 Reverse k-rank query による提案手法の高速化:RkR IQ

本研究では、RR IQ に対して Reverse k-rank query を用いて高速化を行った RkR IQ を提案する。この高速化手法は、Algorithm 1 の処理を行う前に、Reverse k-rank query を用いてクエリ商品を相対的に高く評価するユーザをユーザの好みデータ集合からあらかじめ取得する操作を行い、改良戦略検索の際のランキング更新の際にその絞り込まれたユーザ以外を計算からフィルタリングすることで計算量を減少させる。RkR は R-tree [6] という空間インデックスを利用する。ユーザの好みデータ集合 U と商品集合 P からインデックスを作成し、分枝限定法により効率的にクエリ商品を相対的に高く評価する k 人のユーザを求めることができる。RkR の詳細については [3] を参照。

4. 実　　験

4.1 実 験 環 境

提案手法の有用性を示すために、ランダムに改良戦略を見つけ出し適用する方法と提案手法、そして高速化を行った提案手法についての比較実験を行った。実験環境を以下に示す。

- OS: OS X El Capitan 10.11.6
- CPU: 2.7 GHz Intel Core i5
- メモリ: 16 GB 1867 MHz DDR3
- 開発言語: C++

4.2 比 較 対 象

本研究の比較対象として、RR IQ と RkR IQ の他にランダムに改善戦略を作成するアルゴリズムとのコスト効率、実行時間の比較を行う。ランダムな改善戦略作成のためのアルゴリズムを Algorithm 2 に示す。また、Algorithm 2 内で使われる記号は表 6 と同様とする。

Algorithm 2 Random IQ

Require: P, U, q, τ
Ensure: Improvement strategy s

$q' \leftarrow q$
while $Arank(q') - Arank(q) < \tau$ **do**
  $S \leftarrow \emptyset$
  for $u_j \in U'$ **do**
    Select attribute randomly from q
    Compute score q and one rank above $p \in P$
    Create s with $score_{u_j}(q + s) > score_{u_j}(p)$
    S.add(s)
  end for
  Find $s \in S$ with minimal $Cost_q(s)$
  $q' \leftarrow q' + s$
  Compute $Arank(q')$
end while
Return $s = q' - q$

4.3 実験データ

商品数 $|P| = 10000$、ユーザ数 $|U| = 10000$、$\tau = 50$、次元数 $d = 3$、$k = 1000$ を初期値として、$|P|$、$|U|$、次元数を変量として実験を行った。実験データのデフォルト値、変量を表 7 に示す。また、ユーザの好みと商品の属性値は 0 ～ 1 に正規化を行い、0 に近づくほど良い評価とする。それに伴い、スコアの計算も内積の値がより小さい方が良い評価となり、高いランクがつくことになる。商品集合 P には合成データと実データを使用、ユーザの好みデータ集合 U には合成データを使用して実験を行った。また、P と U の合成データは一様分布と正規分布で生成したデータを使用し、P の実データには民泊予約サイト Airbnb のデータを用いた [10]。この実データは対象となる民泊のレビュー数、全体的な満足度、収容数、ベッドルーム数、価格を含んでいる。

表 7　実験データ設定一覧

データ	デフォルト値	値の範囲
$\|P\|$	10,000	50,000–250,000
$\|U\|$	10,000	50,000–250,000
τ	50	固定
次元 d	3	2–10
k	1,000	固定

4.4 実験データの分布

本研究では、実験の合成データとして一様乱数と正規乱数の 2 つを用いる。一様乱数は、全ての実数が [0, 1] 区間内に均等に分布したものである (図 6)。正規乱数は、平均 0.5・分散 0.2 で作成しており、平均である 0.5 付近にデータが集中した分布である (図 7)。正規乱数では、0 より小さい生成データは 0 とし、1 より生成大きいデータは 1 となるように操作を行った。

4.5 実 験 結 果

実験の基準値は表 7 に示す通りであり、商品 $|P|$、ユーザ数 $|U|$、次元数を変更した時の実行時間の変化を計測した。

一様分布な合成データを用いた実験結果は図 8(a) から図 8(d)

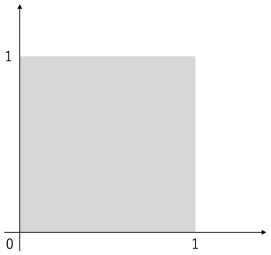


図 6　一様分布のイメージ

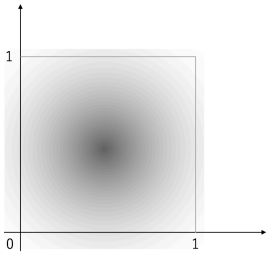
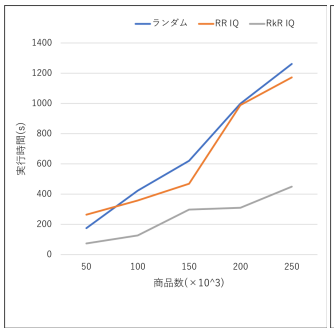
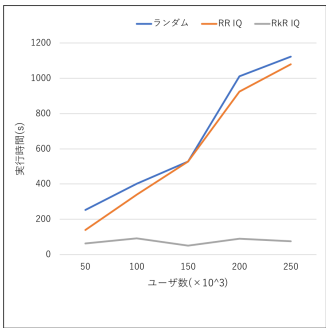


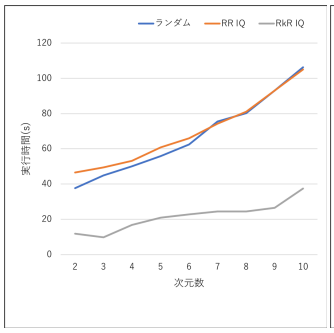
図 7　正規分布のイメージ



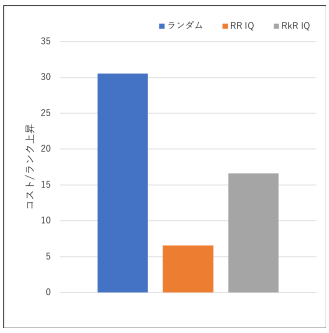
(a) 商品数 $|P|$ による変化



(b) ユーザ数 $|U|$ による変化

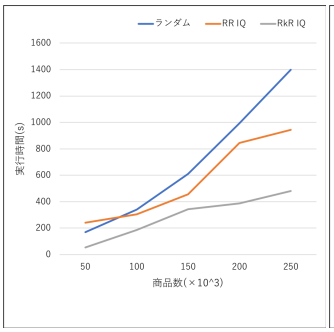


(c) 次元数による変化

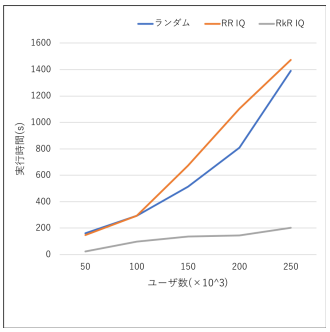


(d) コスト効率

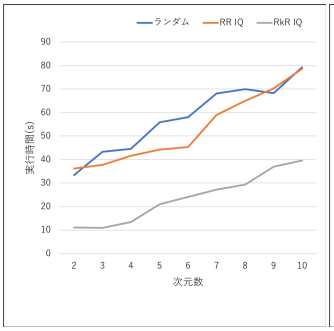
図 8　一様分布データによる実験結果



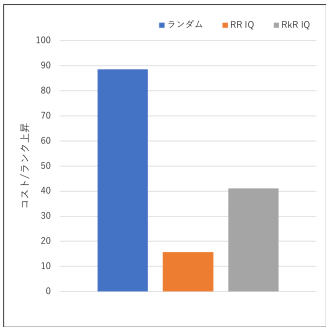
(a) 商品数 $|P|$ による変化



(b) ユーザ数 $|U|$ による変化



(c) 次元数による変化



(d) コスト効率

図 9　正規分布データによる実験結果

に示し、正規分布な合成データを用いた実験結果は、図 9(a) から図 9(d) に示す。合成データによる実験は、商品数 $|P|$、ユーザ数 $|U|$、次元数の変量に対して実行時間がどの様に変化するかをランダムな検索手法・RR IQ・RkR IQ に対して計測し、比較を行った。またコスト効率として (ランク上昇にかかるコスト/ランク上昇量) を計算し、それぞれの手法においての比較を行った。実データについての実験結果は図 10(a) から図 10(b) に示しており、実データに対してのコスト効率と実行速度の計測・比較を行った。

4.5.1　一様分布のデータによる実験結果

図 8(a) は P の値を $P = 50000, 100000, 150000, 200000, 250000$ に変えながら実験を行った結果であり、ランダム手法・RR IQ、と比べて RkR IQ はより高速にクエリ改良の戦略を見つけ出すことができることを示している。図 8(b) は U の値を $U = 50000, 100000, 150000, 200000, 250000$ に変えながら実験を行った結果である。こちらも、ランダム手法・RR IQ と比べて RkR IQ の効果が高いことがわかり、RkR IQ の計算においては、ほぼ一定の値となっている。$|P|$ が変化する場合と $|U|$ が変化する場合で RkR IQ の実行時間に大きく変化するのは、商品数が変数の場合は Reverse k-rank query によるユーザ集合の絞り込みの後でもランキング計算に商品数が関わってくるため実行時間が増えるが、ユーザ数が変数の場合は Reverse k-rank query によるユーザ集合の絞り込みで一定の値に絞り込まれるため、実行速度が一定になり差が生まれたと考えられる。図 8(c) は次元数の変化による実行時間の推移を表している。次元数が増えるほど実行時間は増えているが、RkR IQ の場合は実行時間が抑えられていることがわかる。図 8(d) は一様分布でのランダム手法と RR IQ、また RkR IQ のコスト効率を示すものであり、RR IQ、RkR IQ、ランダム手法の順にコストが高くなっていることがわかる。RkR IQ のコスト効率が RR IQ に対して悪くなっているのは、ユーザデータ集合のフィルタリングによりコスト効率の良い改良戦略の候補の選択肢が少なくなったためだと考えられる。

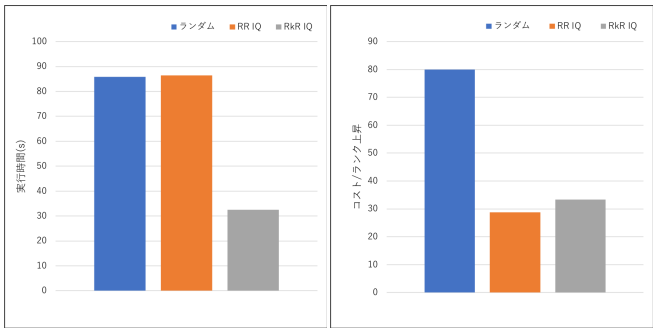
4.5.2　正規分布のデータによる実験結果

図 9(a) は $|P|$ の値を $|P| = 50000, 100000, 150000, 200000, 250000$ に変えながら正規分布の合成データに対して実験を行った結果であり、ランダム手法、RR IQ と比べて RkR IQ はより高速にクエリ改良の戦略を見つけ出すことができることを示している。図 9(b) は $|U|$ の値を $|U| = 50000, 100000, 150000, 200000, 250000$ に変えながら実験を行った結果である。こちらも、ランダム手法、RR IQ と比べて RkR IQ の効果が高いことがわかり、RkR IQ の計算においては、ほぼ一定の値となっている。図 9(c) は次元数の変化による実行時間の推移を表している。次元数が増えるほど実行時間は増えているが、RkR IQ の場合は実行時間が抑えられていることがわかる。図 9(d) はランダム手法、RR IQ また RkR IQ とのコスト効率を示すものであり、RR IQ、RkR

IQ、ランダム手法の順にコストが高くなっていることがわかる。RkR IQ のコスト効率が RR IQ に対して悪くなっているのは、一様分布の場合と同様にユーザデータセットのフィルタリングによりコスト効率の良い改良戦略候補の選択肢が少なくなったためだと考えられる。

4.5.3 実データによる実験結果

図 10(a)〜図 10(b) は実データに関しての実験データを示している。実データは商品集合に民泊予約サイト Airbnb のデータを用い、商品数 $|P| = 21301$、ユーザ数 $|U| = 10000$、$\tau = 50$、$d = 5$、$k = 1000$ を用いて実験を行った。図 10(a) はランダム手法、RR IQ、RkR IQ それぞれの実行時間を示しおり、RkR IQ が他の 2 手法よりも効果が高いことが示されている。また、図 10(b) は実データにおける改良戦略のコスト効率を示しており、ランダムに改良する属性を選ぶ場合より提案手法が約 3 倍コストが少ない改善戦略を提案できることがわかった。



(a) 実行時間　　(b) コスト効率

図 10　実データによる実験結果

5. おわりに

本研究では、先行研究で提案された Top-k query のヒット数を改善の指標とする IQ の問題点を解決するため、平均ランク上昇量を改善の指標とする RR IQ・RkR IQ を提案し、コスト最小化の改良戦略検索について実装・検証を行った。これは、先行研究の問題点である極端なクエリ商品に対して、改良戦略を求めることが困難であるという問題点を解決する。また、実世界の事例により適用しやすくなるように自由な重みづけ関数を利用することを容易にした。そして、ユーザデータ集合 U に対して Reverse k-rank query を適用し計算対象となるデータを減らすことで計算コストを削減した。合成データと実データを用いた実験の結果、コスト効率の良い改良戦略を求めることや高速化が行えることを確認した。今後の課題として、高次元のデータセットを含むさらなる速度改善、実データを用いての最適な重み付け関数の検証などに取り組む必要がある。

文　献

[1] Yang, G., Cai, Y.: Querying Improvement Strategies. EDBT. pp. 294–305, 2017.

[2] Ilyas, I.F., Beskales, G., Soliman, M.A.: A survey of top-k query processing techniques in relational database systems. ACM Comput. Surv. 40(4), 11:1–58, 2008.

[3] Zhang, Z., Jin, C., Kang, Q.: Reverse k-ranks query. PVLDB 7(10), 785796, 2014.

[4] Dong, Y., Chen, H., Furuse, K., Kitagawa, H.: Aggregate Reverse Rank Queries. DEXA 2016, pp. 87–101, September 5–8, 2016.

[5] Dong, Y., Chen, H., Furuse, K., Kitagawa, H.: Grid-Index algorithm for reverse rank queries. EDBT 2017, pp. 306–317, March 21–24, 2017.

[6] Antonin, G.: R-trees: a dynamic index structure for spatial searching. ACM 14(2), pp47–57, 1984.

[7] George, T.H., Gary, P., Stanley, S.: アルゴリズムクイックリファレンス 第 2 版 (黒川 利明ほか訳), オーム社, pp. 350–362, 2016 年.

[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] J. Viner. Cost curves and supply curves. Springer, 1932.

[10] Airbnb data Collection. http://tomslee.net/airbnb-data-collection-get-the-data. Updated: August 7, 2017.