DEIM Forum 2018 D4-3

逆ランク検索による情報改良支援

山下　雅弘† 佐野ひかり† 陳　漢雄†† 古瀬　一隆†† 北川　博之††

† 筑波大学院システム情報工学研究科コンピュータサイエンス専攻　〒 305–8573 茨城県つくば市天王台 1–1–1

†† 筑波大学システム情報系　〒 305–8573 茨城県つくば市天王台 1–1–1

E-mail: †{s1620689,s1411454}@u.tsukuba.ac.jp, ††{chx,furuse,kitagawa}@cs.tsukuba.ac.jp

あらまし　情報検索の技術を応用し、ユーザの嗜好を用いてその嗜好に沿うような商品を検索する研究や、商品を提供する販売者が、その商品を購入してくれそうなユーザを検索する研究が注目を集めている。これらの研究を発展させ、ある 1 つの商品に注目したときに、その商品を一定の予算の中で改良することによって、より多くのユーザから高い評価を得られるように改良するといったような研究も行われている。これらの研究では、ランクと呼ばれる順位付けの概念が用いられている。このような研究の既存の手法では、商品を予算内で改良してもその商品を高く評価するようなユーザが増加しなければ改良されたとみなされない。しかし、そのような場合において実際は個別のユーザからの評価順位は上がっている場合があり、ランクの概念で見たときには適切な評価であるとは言えない。そこで本研究では、ある商品を限られた予算の中で改良したときに、ユーザからのランクの上昇を最大限にもたらすことのできるような改良方法を探すような問題を提案する。一般化すると、商品ではなくオブジェクト（情報）を改良するような問題を考える。この問題を考える上で、オブジェクトのどの属性値をどの程度変更すればランクがどれだけ変わるかを全ての場合において求めるのは NP 困難である。そこで我々は、この問題に対してヒューリスティックに解く手法を提案する。この提案手法では、予算内で考えうる解候補を生成し、逆ランク検索を利用して、ランクが小さいものを複数個取り出すという作業を繰り返し行うことで、網羅的な探索を行い、局所的最適解の可能性を減らすことを行う。そして、この提案手法の効率と精度、実データでの改良提案について実験・考察を行い、その有用性について検証する。

キーワード　Reverse k-Ranks Query, Improvement Query, 情報改良

1. はじめに

ユーザの好みと何かしらのオブジェクト（例：商品）の関係性を用いた検索手法についての研究が近年盛んになっている。それらの研究の中でも、オブジェクトからユーザを検索する手法で、逆 Top-k クエリや逆 k ランククエリ [6] などがある。これらの手法は、メーカー視点の検索モデルとして扱うことができ、潜在的な消費者の発見や商品のマーケティングに応用することができる。

前者の逆 Top-k クエリは、クエリとして与えられたオブジェクトを他のオブジェクトと比べた際に、上位 k 位以内に評価するようなユーザを探す手法である。このとき、上位 k 位以内に評価するようなユーザがいなければ、当然、解となるユーザは 0 人である。また、逆に影響力の強いオブジェクト（すべての属性値が高いオブジェクト）に関しては、殆どのユーザが良いと評価するため、解となるユーザが多くなるという極端な結果が得られてしまう場合がある。後者の逆 k ランククエリは、他のユーザよりもクエリとして与えられたオブジェクトを相対的に好むような上位 k 人のユーザを解として返す。これは、逆 Top-k クエリとは異なり、どのようなオブジェクトに対しても k 人のユーザを検索できる保証がある。

現在、逆 Top-k クエリの性質を利用し、オブジェクトの属性を変えることによってクエリとして与えられているユーザらの好みに合致するようにオブジェクトを改良する Improvement Query [8] と呼ばれるオブジェクト改良の提案を主眼においた研究が存在する。しかし、指摘したように逆 Top-k クエリはオブジェクトの属性値によって極端な結果が得られるため、与えられた予算の中で改良を行っても改良できたかどうかの結果が明確でないという問題がある。

ここで我々が注目したのは、クエリ商品を改良することにより、クエリ商品に対する評価の高いユーザ上位 k 人を得られる逆 k ランククエリの性質である。ここから発展し、商品をどのように改良すれば最も効果的にユーザの評価を高められるかということに対する問題を考えることに興味がある。

本研究では逆 k ランククエリにおいて、クエリとして与えたオブジェクトを改良するときに最小のコストで最大のランク上昇を得られるような改良 s を探すことである。そしてこの問題に対し、提案し、実際に期待した結果を得られるかを検証する。

2. 関連研究

ユーザのオブジェクトに対するランクを基盤とした検索はこれまで広く研究されてきた。ランクは、オブジェクトの評価指標として重要な役割を果たし、それを用いる事によってユーザからの評価の高低を見極めることができる。本研究は、これまで研究されてきた逆ランク検索をはじめとし、Improvement Query などの研究における技術や考え方の多くを取り入れて

いる。そこで、本研究と大きく関わりのある逆ランク検索に関する研究と、Improvement Query の考え方とその手法を紹介する。

逆 Top-k クエリ [1,2]: クエリとして与えられたオブジェクト q を他のオブジェクトセット (P) と比べた際に、上位 k 位以内に評価するようなユーザを全ユーザ (U) から探す手法である。 逆 Top-k クエリを効率化する手法として Vlachou ら [3] は、木構造や、分枝限定法を用いたアルゴリズムを提案した。また、Vlachou らは、[4,5] において、逆 Top-k クエリにおける応用例を多数挙げている。

逆 k ランククエリ [6]: オブジェクト (P) とユーザの好み (U) に関するデータセット、クエリオブジェクト q を与えたときに、すべてのユーザの好みを用いてクエリを含めたすべてのオブジェクトに対してスコアを計算し、スコアによってランク付けを行い、クエリとして与えたオブジェクトを相対的に他のオブジェクトよりも高く評価する上位 k 人のユーザを検索する手法である。また、この問題を拡張し複数のクエリオブジェクトが与えられたときにも対応できるように、[7] が提案されている。

Improvement Query(IQ) [8]: オブジェクトのデータセットと、そのオブジェクトに対する Top-k クエリのセットが与えられたときに、オブジェクトの属性を特定の制限下において調整する。その時の調整の仕方は以下の 2 つの方法がある。

- Min-Cost IQ: 与えられた Top-k クエリのセットの中で、任意の最小の Top-k クエリの解となるように最も効率の良いオブジェクトの改良方法を返す
- Max-Hit IQ: あるオブジェクトに対して改良予算 (B) を与えたときに、その予算内で最大の数の Top-k クエリの解となるようにオブジェクトを改良する

IQ はいずれも、逆 Top-k の性質に着目した商品改良手法であるが、問題として考慮していない点がいくつかある。

（1） Max-Hit IQ において、与えられた予算によっては、改良後の商品が自身の好みに合うようなユーザ数の増加は保証されていない

（2） Min-Cost IQ において、ユーザの好みに合うように商品を調整すると、属性値の変更にかかるコストが非常に高くなる可能性がある

3. 提 案 手 法

本研究では、Improvement Query [8] の中でも Max-Hit IQ の考え方を参考に、与えられた予算内でクエリオブジェクトをどのように改良すれば、最もユーザからのランク上昇をもたらせるかを考える。この問題を Reverse Rank Improvement Query (RRIQ) と呼ぶこととし、この問題が NP 困難であることを証明する。そして、この問題をヒューリスティックに解くための手法、Iterative RkR Extracting 法 (IRkRE) を提案し、詳しく説明してゆく。

以降、用いる記号は表 1 にまとめ、特に断りのない限り表に示した意味として用いることとする。

3.1 問 題 定 義

オブジェクトセット P、ユーザの好みのセット U が与えられ、

表 1　本稿で用いる記号

記号	意味
k	逆 k ランククエリにおけるパラメータ k
d	オブジェクト及びユーザの好みの次元数
P	オブジェクトセット
p	オブジェクト ($p \in P$)
U	ユーザの好みのセット
u	ユーザの好み ($u \in U, \sum_{i=1}^{d} u_i = 1$)
q	クエリオブジェクト ($q = (q_1, q_2, ..., q_d)$)
q'	改良後クエリオブジェクト ($q' = (q'_1, q'_2, ..., q'_d)$)
Q	クエリオブジェクトセット
b	改良予算 ($\geqq 0$)
$RkR(q, P, U, k)$	逆 k ランククエリ関数 ($\geqq 0$)
$c_i(s_i)$	属性改良コスト関数
$Cost(s)$	改良ベクトルに対する改良コスト関数
s	属性改良ベクトル ($s = (s_1, s_2, ..., s_d)$、$\sum_{i=1}^{d} c_i(s_i) \leqq b$)

それらは d 次元空間のベクトルのセットであるとする。$p_i \in P$ なるオブジェクト p_i のそれぞれの次元はオブジェクトの属性値を表すものとし、$u_i \in U$ なるユーザの好みは、各次元の和が 1 となるように正規化されていることとする。クエリとして $q \in P$ なるオブジェクト $q = (q_1, q_2, ..., q_d)$ が 1 つ与えられたときに、逆 k ランククエリによって上位 k 人の q に対するランクを計算し、その結果を $Rank(q) = \{r_1, r_2, ..., r_k\}$ と示す。このランクの合計値を、合計ランク値 $SRank(q) = \sum_{j=1}^{k} r_k$ とする。また、クエリ q を改良し q に改良ベクトル $s = (s_1, s_2, ..., s_d)$ を加えたクエリオブジェクトを q' とする。すなわち、$q_i'^{(j)} = q_i^{(j)} + s_j (1 \leqq j \leqq d)$ であり、s は $\sum_{i=1}^{d} s_i = b$ を満たすものとする。

このとき、逆 k ランククエリを用いて、与えられた予算内でクエリオブジェクトを改良し、ユーザからのランク上昇を最大化する最適化問題は以下のように定義できる。

定義 1. Reverse Rank Improvement Query(RRIQ): クエリオブジェクト $q \in P$, 改良予算 b (ただし, $b \geqq 0$) が与えられたとき、$\{SRank(q) - SRank(q')\}$ が最大となるような改良クエリ q' を求める。ただし、$\sum_{i=1}^{d} f_i(s_i) \leqq b$, $s_i \in \mathbb{R}$ を満たすものとし、かつ $Rank(q) = (r_1, r_2, ..., r_k)$, $Rank(q') = (r'_1, r'_2, ..., r'_k)$ とした時、$1 \leqq i \leqq k$ なるすべての i について $r_i \leqq r'_i$ が成り立つことを前提とする。

RRIQ 問題を携帯電話を商品とした場合の例を用いて説明する。Max-Hit IQ とは異なり、RRIQ の目的は「ランク値」の上昇を最大化するような改良方法を探すことである。説明を簡単にするため、この例において属性改良コストはどの属性値に対しても一律であるとする。表 2 と表 3 は、ユーザの嗜好と商品の属性値を表し、それらにもとづいた携帯電話のランクは表 4 に示されている。 携帯電話 p_2 を自社製品として、改良にかける予算を 1 とする。この例では、p_2 を高く評価するような上位 3 人 (すなわち k=3) のランク値を評価の指標とする。改良後の p_2 を p'_2 としたとき、$p'_2 = \{(7.4, 3.6), (7.1, 3.9), (7.32, 3.68)...\}$ など様々な改良が考えられるが、ここでは説明のため、仮に $p'_2 = (8, 3)$ のときに最大のランク上昇が得られると仮定する。

その時の、ランク値は表 5 に示すとおりである。

表 6 は、Max-Hit IQ と RRIQ の評価の違いを示したものである。k の値は手法によって変えることも当然できるが、この例では同じ k で比べてみる。左が $k = 2$ の場合である。この場合、Max-Hit IQ では改良前と改良後の上位 2 位以内に含まれる人数が変わっていないため p_2' は有益な改良とはみなされない。しかし、RRIQ の結果からわかるように改良前と改良後では、2 から 1 へとランクが上がっている。これは、明らかな改良であるにもかかわらず、Max-Hit IQ では改良とはみなされない。

U	価格	容量
u_1	0.7	0.3
u_2	0.8	0.2
u_2	0.1	0.9
u_2	0.2	0.8
u_3	0.4	0.6

表 2　ユーザの嗜好

P	価格	容量
p_1	7	5
p_2	7	3
p_3	3	6
p_4	1	8
p_2'	8	3

表 3　携帯電話の属性値

	1	2	3	4
u_1	p_1	p_2	p_3	p_4
u_2	p_1	p_2	p_3	p_4
u_3	p_4	p_3	p_1	p_2
u_4	p_4	p_3	p_1	p_2
u_5	p_1	p_4	p_3	p_2

表 4　改良前の携帯電話のランク

	1	2	3	4
u_1	p_2'	p_1	p_3	p_4
u_2	p_2'	p_1	p_3	p_4
u_3	p_4	p_3	p_1	p_2'
u_4	p_4	p_3	p_1	p_2'
u_5	p_1	p_4	p_2'	p_3

表 5　p_2 改良後の携帯電話のランク

	Max-Hit IQ($k = 2$)	RRIQ($k = 2$)
p_2 改良前	2 人	$2 + 2 = 4$
p_2 改良後	2 人	$1 + 1 = 2$

表 6　MaxHit-IQ と RRIQ の結果

3.2　RRIQ 問題が NP 困難であることの証明

RRIQ 問題を、Unbounded Knapsack 問題 (UKP) に対応させ、NP 困難であることの証明を行う。まず、UKP の定義を示し、UKP を RRIQ 問題に還元させる。

定義 2. Unbounded Knapsack 問題 (UKP): $I = \{1, 2, ..., N\}$ を品物の集合とし、各品物 $i \in I$ の重みを w_i、価値を v_i、ナップサックの最大容量を W とし、x_i 個の品物をナップサックに入れるとき、以下の問題を Unbounded Knapsack 問題という。

$$\sum_{i=1}^{N} w_i x_i \leqq W \quad \text{かつ } x_i \geqq 0 \quad \text{を条件としたとき、}$$

$$\sum_{i=1}^{N} v_i x_i \text{を最大化する品物の組み合わせを求める}$$

この UKP は、NP 困難であることが示されている [9]。ここで UKP と RRIQ 問題を対応させ、次の補題を示す。

補題. RRIQ 問題は NP 困難である。

証明. Unbounded Knapsack 問題において、$n = \{1, 2, ..., d\}$ の次元数とし、各次元 $i \in n$ の属性値 q_i、コスト関数を c_i、改良予算を b とし、改良ベクトル属性値を s_i としたとき、RRIQ 問題は以下の数式を用いて表すことができる。

$$\sum_{i=1}^{d} c_i(s_i) \leqq b \quad \text{かつ } s_i \geqq 0 \quad \text{を条件としたとき、}$$

$rank(q + s) - rank(q')$ を最大化する改良ベクトルを求める

ただし、$q = (q_1, q_2, ..., q_d), s = (s_1, s_2, ..., s_d)$

ここで、$rank(q+s)$ の計算複雑度は、UKP における $\sum_{i=1}^{N} v_i x_i$ よりも明らかに複雑である。よって、RRIQ 問題は NP 困難である UKP よりも複雑であるため、RRIQ 問題も NP 困難である。 □

3.3　Iterative RkR Extracting 法

前節の通り、RRIQ 問題は NP 困難である。そのため、最適解を求めることは非常に困難である。そこで、本研究では RRIQ 問題をヒューリスティックに解くことを考える。

本提案手法は、大きく分けて 3 つのパーツによって成り立っている。1 つ目は、クエリオブジェクトを改良するための改良ベクトルを生成し、改良オブジェクト集合を作ること。2 つ目は、有望な改良クエリ集合を抽出すること、そして 3 つ目は、抽出した改良クエリ集合を元に、再度細かく予算を分配し再計算を行うことである。この手法を、Iterative RkR Extracting 法 (IRkRE) と呼ぶ。

ここでは、説明を簡易化するため、$|P| = 1$ とし、改良ベクトルに対する改良コスト関数は $c_i(s_i) = s_i$ とする。IRkRE では、予算をすべての次元に対して網羅的に分配した改良クエリ集合 Q を生成し、その改良クエリ集 Q の中からランク値を返すようにした RkR によって有望な改良クエリ集合 Q' を抽出し、Q' のそれぞれの $q' \in Q'$ について再度改良クエリ集合を作ることを繰り返す事によって、ヒューリスティックに解を得ることを行う。

クエリオブジェクトとして q ($q \in P$) を設定する。入力として、予算 $b(b \in \mathbb{R})$, 予算の分割間隔 δ ($\delta \in \mathbb{R}$, かつ $b \equiv 0 \pmod{\delta}$), ユーザの好み集合 U, 次元数 d, 逆 k ランク検索におけるパラメータ k、反復時に取得する件数 x を受け付けたとき、以下のような手順で処理を行う。

（1） 予算 b と予算の分割間隔 δ を用いて、d 次元空間において網羅的に予算を分配する
改良ベクトル S を生成する。

（2） S を用いて改良オブジェクト集合 Q' を生成する。

（3） Q' について RkR を用いて、$q' \in Q'$ なるオブジェクト q' の上位 k 人のランク合計値を求める。

（4） 3. で求めた合計値から、ランク合計値の小さい x 個のオブジェクトを有望な改良候補とみなし、Q_e' とする。

（5） $q_e' \in Q_e'$ なる q_e' それぞれについて、予算 b と分割間隔 δ を更に一定の割合 ρ で割ったものに対して、余剰分となる予算 $\frac{b}{\rho}$ を q_e' の各次元から引いたものを生成し、再度 1. から 4.

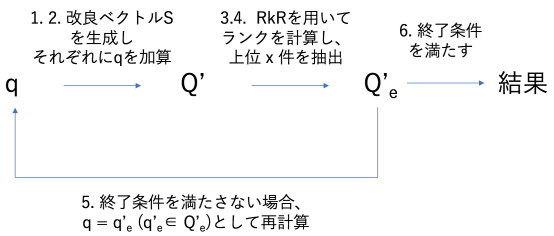

図 1　IRkRE の操作

の計算を行う。ただし、q'_e から余剰予算 $\frac{b}{\rho}$ を引く際に、元のクエリオブジェクト q の属性値を下回らないようにする。ループの終了条件として、任意のループ回数、あるいは任意の桁数までの計算を行った時点で終了することを設定し、結果に Q'_e を加える。

（6） 最終的な改良ベクトル集合を返す。

以上の操作を簡単にまとめたものを、図 1 に示す。

ここからは、順を追って IRkRE の詳細な説明をする。まず、ベクトル S の生成についてみてゆく (1.)。S の生成は、別の問題に置き換えると、値 δ を持つ $\frac{b}{\delta}$ 個の玉を d 個の箱に全て入れる場合におけるすべての組み合わせの結果を求めることと等しい。例えば、次元 $d = 3$、予算 $b = 1$、 予算の分割間隔 $\delta = 0.1$ とすると、

$S = \{(1,0,0),(0.9,0.1,0),(0.9,0,0.1),(0.8,0.2,0),$
$(0.8,0.1,0.1),...,(0,0,1)\}$ のようなベクトル集合を生成することを指す。S を生成するアルゴリズムを、Algorithm 1 に示す。

Algorithm 1 Create Improvement Vector

Input: $\delta, b, d, Record = [0,0,...,0]$ (a d-dimension array) , current_dimension

Output: a set of vector with all possible combination within budget b

(initialize current_dimension ← d only when starting the function)
for ($element = 0$; $(b - element) \geqq 0$; $element$ += δ) **do**
  $Record[current\_dimension - 1] \leftarrow element$
  sum ← accumulate all dimension values lower than $current\_dimension$
  if ($current\_dimension == 2, \&\&(b - sum) \geqq 0$) **then**
    $Record[0] \leftarrow b - sum$
    $result \leftarrow record$
  else if ($current\_dimension == 2, \&\&(b - sum) < 0$) **then**
    $current\_dimension = d$
    $break$
  else
    recursively call self function using ($record$, current_dimension $-1, \delta, b$)
  end if
end for

このアルゴリズムは、第 d 属性に 0 から δ 分ずつ増加させた値を挿入し、第 2 属性から第 $d-1$ 属性までの和を計算させ、その和が予算 b を超えていなければ、$d-1$ 次元目の属性値に対して再帰的に自身を呼び出すことを繰り返す。これを繰り返してゆく中で、$d - 1 = 2$ となり、第 2 属性から第 $d-1$ 属性までの和 sum が b を超えなければ、第 1 属性に $(b$ -$sum)$ を代入し、改良ベクトルの一つとして保存する。もし、第 2 属性から第 $d-1$ 属性までの和 sum が b を超えてしまえば、これ以上計算する必要はないため、一つ外のループに戻り、その次元の属性値に δ を加えて再度計算を行ってゆく。これを最後まで繰り返すことにより、任意の b, δ, d が与えられても、改良ベクトル S を求めることができる。

予算 b と分割間隔 δ の関係は、$b \equiv 0 \pmod{\delta}$ であるので、$\frac{b}{\delta} \in \mathbb{Z}$ である。よって、生成されるベクトル S の大きさは以下の式によって表される。

$$|S| = {}_{(\frac{b}{\delta}+(d-1))}\mathrm{C}_{(d-1)} \tag{1}$$

続いて、改良ベクトル S から改良オブジェクト集合 Q' を生成する (2.)。改良オブジェクト集合 Q' は、クエリオブジェクト q の各属性値に、$s_i \in S$ なる s_i の各属性値を加算したものを $|S|$ 個生成することと等しい。$q'_i \in Q'$ なる q'_i は、$s_i \in S$ なる s_i によって次の式で表される。

$$q'_i = q + s_i (\text{ただし、} 1 \leqq i \leqq |S|) \tag{2}$$

そして、改良オブジェクト集合 Q' が生成されれば、次に RkR に Q' と U を与え、$q'_i \in Q'$ なる q'_i について、q'_i を相対的に高く評価する上位 k 人のランクの合計値を求める (3.) 。そして、その合計値から、ランク合計値の小さい x 個のオブジェクトを有望な改良候補とみなし、有望改良候補集合 Q'_e を生成する (4.)。

RkR は本来、ある 1 つのオブジェクト q を他のオブジェクト $p \in P$ よりも相対的に高く評価する k 人を検索する手法である。このとき、返される結果をユーザではなく、そのユーザのランクを返すように出力を変えることによって、上位 k 人のランクが求まるようになる。式として表すと、$RkR(q'_i, Q', U, k)$ となる。ここで重要なのは RkR で用いられる P として Q' を与え、q に $q'_i \in Q'$ を与えていることである。ここで P として Q' を与えることによって、改良オブジェクト集合 Q' のなかで、q' がどの順位にいるかがわかるのである。この操作を全ての $q' \in Q'$ に行うことにより、Q' における最良の改良オブジェクトを探すことが可能となる。そして、Q' においてランクの合計値の小さい x 個のオブジェクトを有望な改良候補集合 Q'_e とする。

最後に、$q'_e \in Q'_e$ なる q'_e それぞれについて、再度上記で説明した計算を終了条件に見合うまで繰り返し行う (5.)(6.)。ただし、繰り返しの際には、b と δ を一定の割合 ρ で割り、全ての $q'_e \in Q'_e$ なる q'_e の各属性値から $\frac{b}{\rho}$ を引いたオブジェクトを生成し、Q'_e を更新する。

例えば、$b = 1, \rho = 10$ であった場合、クエリオブジェクト $q = (4.0, 4.6, 5.4)$ が与えられたときに、改良オブジェクト $q'_e = (4.0, 5.0, 6.0)$ が生成されたとする。$\frac{b}{\rho} = 0.1$ であるので、Q'_e の

可能性として、$\{(3.9, 5.0, 6.0), (4.0, 4.9, 6.0), (4.0, 5.0, 5.9)\}$ の 3 つが考えられる。しかし、第 1 属性に注目した時、元のクエリオブジェクトは 4.0 であり、元のクエリオブジェクトよりも属性値が下回っている要素である $(3.9, 5.0, 6.0)$ は更新の対象外であるため、結果として $Q'_e = \{(4.0, 4.9, 6.0), (4.0, 5.0, 5.9)\}$ となる。この作業を行う事によって、常に予算 b の中で改良が行われることが保証され、かつ元のクエリの状態を逸脱したようなオブジェクトを生成することを防ぐことができる。

終了条件として、任意のループ回数を超えたとき、あるいは任意の桁数までの計算を行った時点までとすることにより、検索する側が任意の精度、あるいは任意の時間内で計算を終了することを保証する。

以上の内容を満たすアルゴリズムを Algorithm2 に示す。

Algorithm 2 Iterative RkR Extracting

Input: $b, \delta, d, U, P, q, k, x$
Output: a set of top-x improved q

```
S ← CreateImprovementVector(d, δ, b) ... Algorithm1
for each s ∈ S do
    q' ← q + s
    Q' ← q'
end for
for u ∈ U do
    for q' ∈ Q' do
        score ← q' · u
        index ← {q', u, score}
    end for
end for
for q' ∈ Q' do
    SRank(q') ← modifiedRkR(q', score, k)
end for
Sort SRank(q') by smallest
result ← extract top-x q' with regard to SRank(q')
return result
```

4. 実験・考察

本実験では、ユーザの嗜好データ及びオブジェクト集合データとして合成データを利用し、ユーザの嗜好データには一様分布および二項分布のデータを、オブジェクト集合データとして一様分布のデータを用意した。オブジェクトの取りうる値の範囲は、[0,10] とする。

本実験は、以下のマシン条件のもとにおいて行われ、実装に用いた言語は C++である。

- CPU: Intel Core i5 2.6GHz
- メモリ: 8GB
- OS: Mac OS X Yosemite 10.10.5

表 7 は、実験にて用いるパラメータについて、それぞれの既定値およびどの程度の範囲で計算させたかを表している。改良対象オブジェクト q は、$|P|$ から 1 つランダムなオブジェクトを選択するものとする。

計算速度の変化の実験として、次元数 d の変化、ユーザの好みの数 $|U|$ の変化、そして予算 b と分割間隔 δ の変化による処理を 10 回行い、その平均処理速度を割り出し、その結果を 4.1 節に記す。

パラメータ	既定値	範囲
次元数 d	3	2 - 5
ユーザの嗜好集合 $\|U\|$	50,000	10,000 - 100,000
オブジェクト集合 $\|P\|$	1,000	1,000 - 5,000
改良対象オブジェクト q	1	-
予算 b	1	1 - 10
分割間隔 δ	0.1	$\frac{b}{10}, \frac{b}{20}, \frac{b}{50}, \frac{b}{100}$
取得ユーザ数 k	$\frac{\|U\|}{100}$	-
ループ時の抽出件数 x	$\log_d \|Q_i\|$	-

表 7　実 験 設 定

4.1 計算コスト

IRkRE 法において、1 度のループにかかる処理時間を調べた結果を以下に示す。

図 2 は、ユーザ数 $|U|$ が増えたときの処理時間の変化を示したものである。次元数に関わらず、処理時間はユーザ数が増加するに従って線形的に増加していることがわかる。図 3 は、オブジェクトの次元数が増えたときの処理時間の変化を示したものである。次元数が増えるごとに、計算時間が増えていることがみてとれる。特に 4 次元と 5 次元の場合の差が顕著であり、より高次元になればなるほど処理時間は増大すると考えられる。

図 4 は、予算 b とその分割間隔 δ の関係による処理時間の変化を示したものである。第 3 章に記述したとおり、IRkRE 法において改良ベクトル集合 S の大きさは $|S| = {}_{(\frac{b}{\delta}+(d-1))}\mathrm{C}_{(d-1)}$ によって表される。したがって、$\frac{b}{\delta}$ の値が大きくなればなるほど、改良ベクトル集合 S も大きくなることになり、相対的に改良オブジェクト候補 Q が増加するため、処理にかかる時間が増大すると考えられる。しかし、実験の結果から分かるように $\frac{b}{\delta}$ の値が増加しても処理時間は緩やかに増加しているが見て取れるため、より細かい粒度で計算を行ったとしても処理時間はある程度抑えられると考えられる。

4.2 精　　度

精度の実験として、様々な q に対し、$\delta = 0.1$ のときに 2 度ループさせたとき、及び $\delta = 0.01$ のときに 1 度ループさせたときに、それぞれ上位 τ の改良候補を比較する。改良候補を Q^α, Q^β と記すとき、比較は次の数式によって行う。

$$\epsilon = \sum_{q_i \in Q^\alpha} \min_{q_j \in Q^\beta} ||q_i q_j||_1$$

ただし、$||uv||_1$はベクトル u, v の L1 距離を表すものである

表 8 は、様々な τ を設定したときに平均的にどれだけの相違が発生するかを表したものである。

この結果から、τ に依らず相違が一定して低い状態を保っている事がわかる。これは、τ の状態によらず一定の解の出力を保っていることを示している。つまり、IRkRE 法が低い粒度であっても、処理時間の短い計算を複数回行うことによって総合的な処理時間を短縮しつつ、ヒューリスティックに有効な解を得られることを示している。また、同時に高い粒度であっても現実時間内で解を得られることも保証できている。

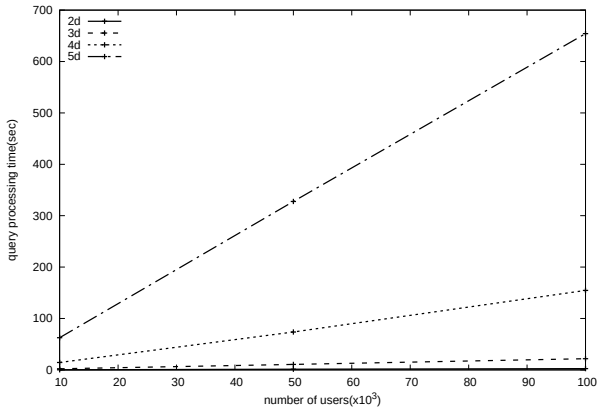

図 2　ユーザ数 $|U|$ による処理速度の変化

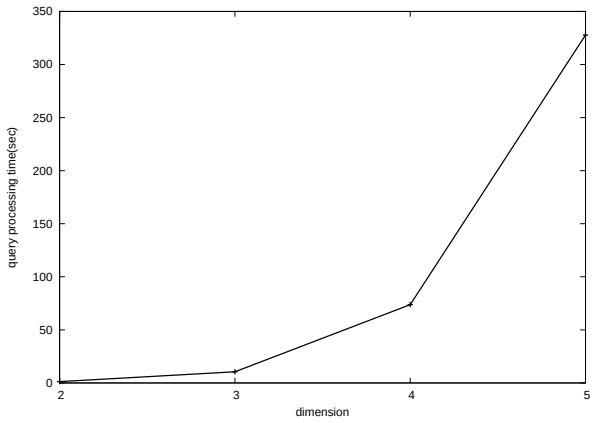

図 3　次元数 d による処理速度の変化

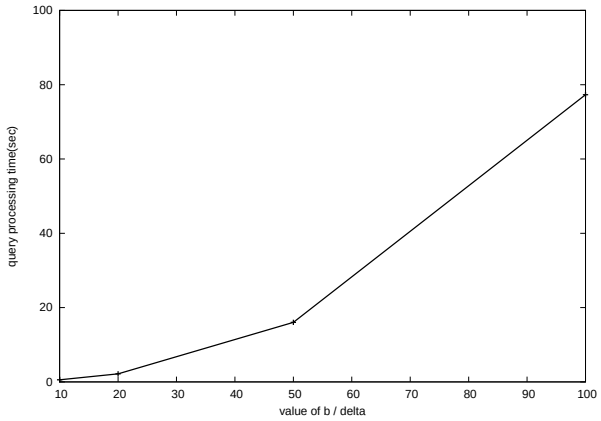

図 4　予算とその分割間隔の関係による処理速度の変化

τ	ϵ	$\delta = 0.1$ の実行時間	$\delta = 0.01$ の実行時間
10	0.01	3.325	76.322
20	0.007		
30	0.006		
50	0.0048		
100	0.0038		

表 8　実 験 設 定

5. 実データを用いた改良提案

実データを用いた実験として、ランダムにオブジェクトを 1 つ選択し、そのオブジェクトに対してどのような改良を提案するかを調べる。実データにおけるコスト関数を表 9 に示す。ループは 2 回に設定し、属性値は互いに影響を及ぼすことがないと仮定する。表中の δ_i は、i 次元の属性値にかけた予算を表す。予算は $b = 1$ とし、各属性値に対して予算を割り振ったときに、s_1 は最大で 5、s_2 は最大で 2、s_3 は最大で 2, s_4 は最大で-100 の変更が発生するように制限をかけた。また、ランクを決定する際に必要なスコアを計算するスコア関数に対して、第 4 属性の価格のみ値が高くなるとスコアが低くなるような調整を施した。

表 9　実データにおけるコスト関数

属性値	コスト関数
満足度 (s_1)	$\delta_1 = \log_{11}(2 * s_1 + 1)$
収容可能人数 (s_2)	$\delta_2 = \frac{s_2}{2}$
ベッドルーム数 (s_3)	$\delta_3 = \log_2(s_3 + 1)$
価格 (s_4)	$\delta_4 = -\frac{s_4}{100}$

図 5 は、実際にあるオブジェクトに対してどのような改良を提案するかを示したものである。1 行目は、オブジェクトが全 2,000 件のデータの中でランクがいくつかを表し、その時の全ユーザ 10,000 人からのランクの合計値を表したものである。2 行目は、改良後のオブジェクトに関する同様の情報である。7 行目以降は、改良後のオブジェクトに関してどのような候補があるかをランクが高い順に示したものである。このオブジェクトの場合、施設を利用した際の満足度を上げるような改善を行うことができれば以前よりもユーザからの評価が高くなる改良を提案していると同時に、一泊あたりの価格を下げることもユーザからの評価が高くなるような提案を示している。

6. 結　　論

本稿では、我々の研究分野である逆ランク検索と情報改良に関して、ランクを用いて改良の度合いを評価する新たな考え方を取り入れた情報改良問題である RRIQ 問題を提案した。そして、RRIQ 問題が NP 困難であることを証明し、この問題を現実時間内でヒューリスティックに解くような手法である IRkRE 法を提案した。実験において、IRkRE を実装しその計算速度と精度について、合成データを用いて確認した。また、実データを用いて IRkRE 法を用いた際にどのような改良を提案するかについての実験も行い、その有用性を確かめた。

今後の課題として、以下のような改善が考えられる。1 つ目は、高次元に対応した効率化を考えることである。現在、次元数が増えるごとに計算時間が大きく増加しているため、属性値の多いオブジェクトの改良に非常に時間がかかるという問題点がある。2 つ目は、属性値の増減を考慮したときに最大限の改良ができるようなアルゴリズムを考案することである。現在、属性値は増加方向の単調性を用いて計算を行っている。もし、減少方向も考慮に入れて考えれば、オブジェクトのある属性値を減らし、そのコスト分を別の属性値に移し替えるようなオブジェクト改良を考えることにつながる。3 つ目は、RkR の上位 k 人という制約にこだわらず、全ユーザに対して、q、q' を与えたときにランクが上昇したユーザ、ランクが下降したユーザ、ランクが変わらなかったユーザなどに分類し、これらのユーザをまとめてクラスタ化し、各クラスタについてどのような属性改良を施した方がより有益であるかどうかを考えるようなアダプティブなクエリ改良法を考えることも課題として挙げられる。

文　　献

[1] Vlachou, A., Doulkeridis, C., Kotidis, Y., et al.: Reverse top-k queries. In: ICDE, pp. 365-376 (2010)

[2] Vlachou, A., Doulkeridis, C. , Kotidis, Y., et al.: Monochromatic and bichromatic reverse top-k queries, pp. 1215-1229 (2011)

[3] Vlachou, A., Doulkeridis, C., Kotidis, Y., et al.: Branch-and-bound algorithm for reverse top-k queries. In: SIGMOD Conference, pp. 481-492 (2013)

[4] Vlachou, A., Doulkeridis, C., Kotidis, Y.: Identifying the most influential data objects with reverse top-k queries, pp. 364-372 (2010)

[5] Vlachou, A., Doulkeridis, C., et al.: Monitoring reverse top-k queries over mobile devices. In: MobiDE, pp. 17-24 (2011)

[6] Zhang, Z., Jin, C., Kang, Q.: Reverse k-ranks query. PVLDB 7(10), 785-796 (2014)

[7] Y. Dong, H. Chen, K. Furuse, and H. Kitagawa. Aggregate Reverse Rank Queries LNCS, 9828:87- 101 (DEXA 2016)

[8] Guolei Y., Ying C.: Querying Improvement Strategies. In: EDBT, March 21-24, 2017, Venice, Italy

[9] Paul E. Black, "unbounded knapsack problem", in Dictionary of Algorithms and Data Structures [online], Vreda Pieterse and Paul E. Black, eds. 27 September 2013. https://www.nist.gov/dads/HTML/unboundedKnapsack.html (閲覧日: 2017 年 12 月 28 日)

[10] Tom Slee (2017-8-7). Airbnb Data Collection: Get the Data. http://tomslee.net/airbnb-data-collection-get-the-data (閲覧日: 2017 年 12 月 28 日)


```
Original Rank: 914, SRank: 9178648
Best Rank: 240, Best SRank: 3619527

Original object
0,10,4,242

Result (Order by highest rank)
5, 10, 4, 242
4.37858, 10, 4, 237
0.0636891, 10, 4, 147
0.0636891, 10.1, 4, 152
3.82736, 10, 4, 232
0.0636891, 10.2, 4, 157
0.0636891, 10, 4.03526, 152
0.135491, 10, 4, 152
0.0636891, 10.1, 4.03526, 157
0.135491, 10.1, 4, 157
0.0636891, 10.3, 4, 162
3.33843, 10, 4, 227
0, 10.5, 4, 167
0.216438, 10, 4, 157
0.0636891, 10, 4.07177, 157
0.0636891, 10.2, 4.03526, 162
0.135491, 10, 4.03526, 157
0.135491, 10.2, 4, 162
0, 10.4, 4.03526, 167
0.216438, 10.1, 4, 162
2.90474, 10, 4, 222
0.0636891, 10.1, 4.07177, 162
0.307697, 10, 4, 162
0, 10.3, 4.07177, 167
0.0636891, 10.4, 4, 167
0.135491, 10.1, 4.03526, 162
0, 10.2, 4.10957, 167
0.0636891, 10, 4.10957, 162
0.216438, 10, 4.03526, 162
0.0636891, 10.3, 4.03526, 167
0.135491, 10, 4.07177, 162
0.135491, 10.3, 4, 167
0.41058, 10, 4, 167
2.52005, 10, 4, 217
0, 10.1, 4.1487, 167
0.216438, 10.2, 4, 167
0.307697, 10.1, 4, 167
0.0636891, 10.2, 4.07177, 167
0, 10.6, 4, 172
0.135491, 10.2, 4.03526, 167
0, 10, 4.18921, 167
0, 10.5, 4.03526, 172
0.526568, 10, 4, 172
...
```

図 5　実データを用いた改良提案の結果