Max/Min Aggregate Reverse Rank Queryに対する効率的な検索手法

千田　俊[†]　董　于洋[†]　陳　漢雄[†]　古瀬　一隆[†]

† 筑波大学　〒 305-8571 茨城県つくば市天王台 1 丁目 1 − 1

E-mail: †s1513107@u.tsukuba.ac.jp, ††tou@dblab.is.tsukuba.ac.jp, †††{chx,furuse}@cs.tsukuba.ac.jp

あらまし　近年、一つの商品がクエリとして与えられたとき、その商品を最も好む k 人のユーザーを求める Reverse rank query という検索が注目を集めている。その発展として、複数の商品をバンドルして販売するケースの検索方法、Aggregate Reverse Rank Query（ARkR）が提案されている。ARkR では集約関数 Sum についての検索アルゴリズムが提案されている。本研究では集約関数 Max/Min について効率的な検索手法を提案する。先行研究の手法と比較した結果、本手法は従来の手法より 3 倍以上の処理速度が確認された。

キーワード　Aggregate Reverse Rank Query, ARkR, 問い合わせ処理, データマイニング

1. はじめに

Top-k Query [1] [2] と Reverse Rank query [3] はマーケティングにおいて 2 つの有効なアップローチである。Top-k Query はユーザーの視点からアプローチし、一人のユーザーの好みをクエリとして与え、そのユーザーに対して各商品のスコアを計算して商品のランク付けを行う。ユーザーが最も好みそうな k 個の商品を解として返す。一方、Top-k Query の逆の意味で Reverse rank query は販売者の視点からアプローチし、商品のレートからその商品を最も好みそうなユーザーを探索するクエリである。一つの商品をクエリとして与え、ユーザーそれぞれの Top-k ランク付けを行い、クエリとして与えられた商品を最も上位にランク付けする k 人のユーザーを解として返す。[4] [5] [6] [7] このクエリを用いて、商品の販売者は潜在的な顧客を特定してより効率なマーケティングが可能になる。

表 1〜表 3 では、Reverse Rank Query の k=1 の例を示している。表 1 では 3 人のユーザー（u1〜u3）がパソコンに関する好みを示し、好みは性能、価格、デザインに分けて合計 10 とする。表 2 では 5 つのパソコン (p1〜p5) について性能、価格、デザインのレートを示している。表 3 では Reverse rank query の結果を示している。ユーザーによる商品のスコアはユーザーの好みベクトルと商品のレートベクトルの内積で計算される。また、スコアは小さい順でランク付けされる。例えば、ユーザー u1,u2,u3 による p1 のランキング順位はそれぞれ 1,4,3 となり、u1 による p1 のランキング順位が最も高いので、p1 の r-1rank の解は u1 となる。

表 1　ユーザーの好み

ユーザー	性能	価格	デザイン
u1	1	3	6
u2	5	1	4
u3	4	3	3

Reverse rank Query は一つの商品に対する問い合わせであるが、実際マーケティングでは複数の商品をバンドルして販売するパターンも多く存在する。そういう販売形式を支援するため、Aggravate Reverse Rank Query（ARkR）が提案された [8]。

表 2　パソコンのレート

パソコン	性能	価格	デザイン
p1	9	1	6
p2	5	2	7
p3	6	4	8
p4	7	8	8
p5	3	3	9

表 3　パソコンによるユーザーのスコアと r-1rank

パソコン	u1	u2	u3	u1 による ランク	u2 による ランク	u3 による ランク	r-1rank
p1	48	70	57	1	4	3	u1
p2	53	55	47	2	2	1	u3
p3	66	66	60	3	3	4	u1
p4	79	75	76	5	5	5	u1
p5	66	54	48	4	1	2	u2

ARkR は複数の商品をバンドルした商品セットをクエリとして与え、その商品セットを最も上位にランク付けする k 人のユーザーを返す。ARkR の評価方法は複数定義されているが、表 4 では商品セット内の最もランクが高い商品を全体のランクとして評価した結果を示している。

表 4　AR-1rank

商品セット	u1 による max ランク	u2 による max ランク	u3 による max ランク	AR-1rank
p1, p3	1	3	4	u1
p2, p3	2	2	1	u3
p2, p4	2	2	1	u3
p4, p5	4	1	2	u2

1.1 問題定義

商品のデータ集合を P、ユーザー好みのデータ集合を U、各商品 p ∈ P が d 次元のベクトルで d 個のレートを持ち、商品 p の各レートは p[1], p[2], ..., p[d] とする。そして、各ユー

ザー u ∈ U が d 個の好みをもつ d 次元のベクトルであり、ユーザーの好みは u[1],u[2],...u[d] として、$\sum_{i=1}^{d} u[i] = 1$ とすると、Reverse Rank Query は以下に定義される。

定義 1. *(rank(u, q))*：商品のデータ集合を P、ユーザーの好みを u、クエリとしてを q 与えた場合、u による q のランク付けは $rank(u,q) = |S|$ である。ただし、$S \subseteq P$ かつ$\forall\ p_i \in S : f(u,p_i) < f(u,q) \wedge \forall\ p_j \in (P - S) : f(u,p_j) \geqq f(u,q)$ が成り立つ.

定義 2. *(reverse k-ranks query)*： 商品のデータ集合を P, ユーザーの好みデータ集合 U、正整数 k、クエリ商品 q を与えた場合。*Reverse k-Rank Query* はデータ集合 S を返す。ただし、$S \subseteq U, |S| = k$ であり, $\forall\ u_i \in S, \forall\ u_j \in (U - S) : rank(u_i,q) \leqq rank(u_j,q)$ が成り立つ.

商品集合をクエリとした問い合わせを解決するため、ARkR では以下の定義を行っている。

定義 3. *(Aggregate Reverse Rank Query, AR-k)*：商品のデータ集合を P, ユーザーの好みデータ集合 U、正整数 k、クエリされた商品集合 Q を与えた場合。*ARkR* はデータ集合 S を返す。ただし、$S \subseteq U, |S| = k$ 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.

ARkR は商品集合をクエリとした問い合わせ問題に 3 つの評価方法を提案した。それぞれは下記の Sum、Max、Min の 3 つ集約関数に対応する。

- $Sum : ARank(u,Q) = \sum_{q_i \in Q} rank(u,q_i)$
- $Max : ARank(u,Q) = Max_{q_i \in Q}\{rank(u,q_i)\}$
- $Min : ARank(u,Q) = Min_{q_i \in Q}\{rank(u,q_i)\}$

1.2 モチベーション

集約関数 Sum は、商品セット Q のぞれぞれの商品のランクの和をその商品集合の評価とする。実際の買い物の場合、ユーザーが商品セットのそれぞれの商品について考えてた上、商品セットの全体価値が他の商品セットより良いかどうかで判断する購買パターンに対応する。一方、ユーザーは商品セットの中にとても好きな商品が入っていればそのセットを全部買う、嫌いの商品がなければその商品を買うというパターンも多く存在する。例えば、福袋に一つとても気に入る商品があれば、或いは全然使えない商品がなければユーザーがその福袋を買ってよかったと思う。そういうパターンに対応するのは集約関数 Max と Min である。集約関数 Max は商品集合 Q の中の最もランクの高い商品のランクを商品集合の評価とする。ユーザーが商品セットの中一つとても好きな商品があれば、その商品セットを買うという購買パターンに対応する。集約関数 Min は Max の逆のアプローチをして、ユーザーが商品セットの中嫌いな商品がなければその商品集合を買うという購買パターンに対応する。

先行研究では Sum に対して 2 種類の効率的な検索手法が提案されたが、それらの手法は Max/Min に応用すると非効率である。そのため、本研究では Max/Min に対する効率的な検索手法を提案する。

2. 関 連 研 究

先行研究では ARkR の集約関数 sum が注目され、TPM と DTM という 2 種類の手法が提案された。これらの手法は R 木を用いて、商品集合をレートの近い物でクラスタリングしてインデックス化を行う。インデックスされた商品集合の極大値と極小値を用いて必要のない計算を探索から除外し、計算コストの削減を図った。

2.1 TPM

TPM では R 木を用いて類似する商品の最小包囲矩形（MBR）を作成し、商品集合 P をインデックス化する。そして、計算する必要がない MBR を探索から除外することで計算コストを削減する。

図 1 では TPM の 2 次元の場合の例を示す。TPM ではまず Q に対して MBR を作成し、その MBR の右上の点を Q_{up}、左下の点を Q_{low} として、各ユーザーの好みベクト $u_i \in$ U から、Q_{up} と Q_{low} に垂線を引く。それによって、図 1 のように 2 次元空間は BelowQ、InQ、AboveQ の 3 つのサブ空間に分けられる。この例の場合、e_2 は BelowQ に、e_3 は InQ に含まれ、e5 は AboveQ に含まれる。また、e_1、e_4 のは複数のサブ空間に含まれる。この場合、BelowQ に含まれるデータの u によるランク付けは必ず Q に含まれるデータの u によるランク付けより上位になり、AboveQ に含まれるデータの u によるランク付けは必ず Q に含まれるデータの u によるランク付けより下位になる。この性質を用いて、TPM は BelowQ に含まれるデータをランキングの下限として数えあげて、AboveQ に含まれるデータを計算せずに探索から除外することで、計算のフィルタリングを行う。したがって、BelowQ と AboveQ の範囲が大きくなるほどフィルタリングの効果が良い。さらに、InQ に含まれるデータをそれぞれの q に対するランクを計算し、BelowQ で求められたランキングの下限に加算することで、商品セット Q の ARrank を算出する。

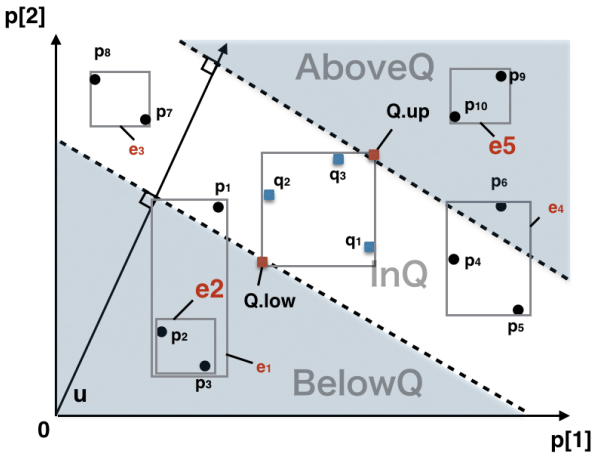


図 1　TPM の 2 次元のイメージ

TPM は初期化を行う必要がある。まずはユーザーの好みベクトル集合 U の最初の k 個 U1 = u[1]、u[2]、...u[k] ∈ U を取り出し、その k 人のランキングを作り、heap の初期値とする。

その中の最も順位が低い人のランク heap[k-1] を minRank とする。そして、残った人 U2 = U - U1 を上記フィルタリングのルールを用いて各 u による Q のランクを計算する。u による Q の ARrank が minRank より大きい場合、u を heap に挿入し、minRank を更新する。u による Q の ARrank が minRank より小さい場合、次の u について計算する。U2 に含まれるすべてのデータについて計算後、heap に残った k 人のランキングが最終的な ARrank となる。

2.2 DTM

DTM は TPM を発展したアルゴリズムである。TPM は R 木を用いて類似する p を分類し、計算する必要がない MBR を探索から除外して、u による Q のランク付けを計算するコストを大幅に削減できたが、各 u については逐次計算する必要があるため、$|U|$ の数が大きくなるほどその効率が低くなる。DTM では U 対しても R 木を用いて類似する u を MBR で分類し、MBR ごとに探索を行った。一回に複数の u をフィルタリングすることで計算効率を更に向上させた。

2.3 TPM、DTM の効果

表 5 ではネーティブ手法、TPM、DTM の計算量を示している。全体的にネーティブ手法は最も計算コストが高く、$O(|P| \cdot |U|)$ が必要だった。TPM と DTM はインデックス化するための時間がかかるが、ランキングを計算するためのコストが激減した。

表 5　ネーティブ手法、TPM と DTM の効率まとめ

手法	インデックス	ランキングを計算コスト	インデックス化コスト
NA	なし	$O(\|P\| \cdot \|U\|)$	O(k)
TPM	RtreeP	$O(\|U\| \cdot log\|P\|)$	$O(log\|P\|)$
DTM	RtreeP、RtreeU	$O(log\|U\| \cdot log\|P\|)$	$O(log\|U\| + log\|P\|)$

2.4 TPM、DTM の課題

TPM、DTM はクエリ集合 Q を用いて、商品集合 P の空間を 3 つのサブ空間に分割し、BelowQ と InQ に含まれる商品をフィルタリングする事によって計算コストを削減した。しかし、実際の販売では評価の人気商品とそうでもない商品のようなレートが大きく異る商品をバンドルして販売するケースも多く存在する。そういった場合は図 2 に示したように商品集合 Q の中各 q のレートが大きく異り、BelowQ と InQ の領域が小さくなりフィルタリングできる計算も大幅に下がってしまう。

3. 提案手法

本研究では、クエリ集合 Q に対して前処理を行うことで探索の効率化を図る。集約関数 Sum で評価する場合、各商品のランクの和を ARrank 評価とするため、商品集合に含まれる各データのランクを調べる必要がある。しかし、Max/Min の集約関数は集約関数 Sum と違い、ユーザーによって最もランク付けが高い商品或いは最もランク付けが低い商品のランクが分かれば ARrank を算出できる。この特性を用いて、本研究では商品集合 Q の境界点となるデータを絞る事に注目した。極値以外の商品を除外することによって $|Q|$ を減らし、TPM や DTM で

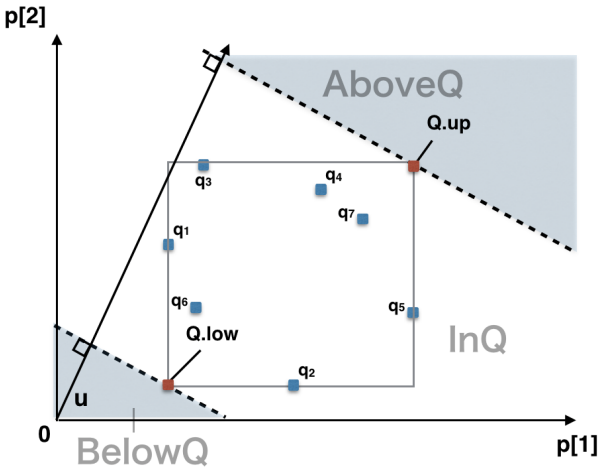


図 2　TPM、DTM が非効率になる Q

検索から除外できるサブ空間を拡大して探索の効率化を行う。

図 3 では 2 次元の Q に対する前処理を示している。ただし、使用する集約関数は Max である。クエリ集合 Q が与えられた時、まずは Q の凸包 $CH(Q)$ を作成する。$CH(Q)$ に含まれるデータはすべて Q の境界点なので、u によるランク付けする時の最大、最小になる q は必ず $CH(Q)$ に含まれる。したがって、$ARrank(Q) = ARrank(CH(Q))$ が成り立つ。この例の場合、u によるランク付けが最も高い q は必ず $CH(Q) = \{q_1, q_2, q_4, q_5, q_9, q_{10}\}$ に含まれる。

次に、TPM や DTM を適用する場合と同様に Q の最小包囲矩形 $MBR(Q)$ を作成する。この場合 Q_{low}は必ず $MBR(Q)$ の中の u によるランク付けする時の下限であり、Q_{up}は必ず集合 $MBR(Q)$ の中の u によるランク付けする時の上限である。従って、Max/Min 集約関数で ARrank を求める場合、u によるランク付けが最も高い q は Q_{low}/Q_{up} の付近に存在すると予想できる。そこで、それぞれの次元の Q_{low} に最も近いデータを求めて $Q\prime$ とし $MRB(Q\prime)$ を作成すると、u によるランク付けが最も高い q は必ず $MBR(Q\prime)$ に含まれる。したがって、$ARrank(u, Q) = ARrank(u, MBR(Q\prime))$ が成り立つ。この例の場合、Q_{low} を原点として p1 軸と p2 軸の方向に探索し、それぞれの軸で出会った一つ目の q を用いて $MBR(Q\prime)$ を作成する。u によるランク付けが最も高い q は必ず $MBR(Q\prime) = \{q_1, q_2, q_3, q_5, q_6\}$ に含まれる。

以上より、u によるランク付けが最も高い q は必ず $CH(Q)$ に含まれるデータであり、かつ、$MBR(Q\prime)$ に含まれるデータである。したがって、u によるランク付けが最も高い q はクエリ集合 $Q_m = CH(Q) \cap MBR(Q\prime)$ に含まれ、$ARrank(u, Q) = ARrank(u, Q_m)$ が成り立つ。この例の場合、$Q_m = \{q_1, q_2, q_5\}$ となり、$Q_m = \{q_1, q_2, q_5\}$ を用いて求めた ARrank の結果は ARrank(u,Q),$Q = \{q_1, q_2, ..., q_{10}\}$ の結果である。

図 3 に示される $AboveQ$ は前処理を行わずに TPM を用いて u のランクを計算する時、探索から除外できる領域であり、$AboveQ\prime$ は前処理を行なってから TPM を用いて u のランクを計算する時、探索から除外できる領域である。前処理を加えることで、探索から除外できる領域が大幅に拡大され、各 u に対

して計算すべき q のランクの数も減らされた。省略できる領域が拡大することで、TPM や DTM の Q の分布が広いと検索効率が悪くなる課題が解決され、計算速度が大幅に上がる。

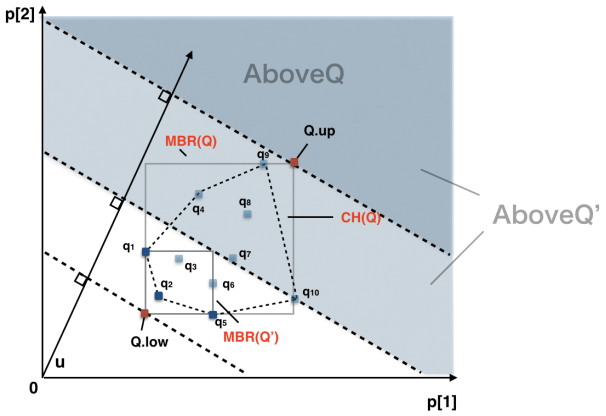


図 3　Q の前処理

上記手法の擬似コードをアルゴリズム 1 に示す。入力を商品集合 Q として、出力を前処理を加えた後の商品集合 Q_m とする。1-4 行目では使用する変数の宣言を行う。1 行目で $Q\prime$、Q_m を空配列として宣言する。2-3 行目で、Q の凸包 CH(Q) と Q の最小包囲矩形 $MBR(Q)$ を作成し、3 行目で $MBR(Q)$ の左下の点 Q_{low} の宣言を行う。

5-13 行目では、各次元の Q_{low} に最も近い点を探して $Q\prime$ に保存し、すべての次元で探した後 $Q\prime$ 内のデータを用いて MBR($Q\prime$) を作成する。

15-27 行目では MBR($Q\prime$) の範囲内にあり、かつ、CH(Q) に含まれるデータを求める。15-16 行目で MBR($Q\prime$) の範囲を決める左下と右上の点をそれぞれ $range_{low}$、$range_{up}$ に代入する。

そして、17-28 行目で CH(Q) 内のすべてのデータを取り出し、それぞれ $range_{low}$、$range_{up}$ と比較し、$range_{low}$より小さい或いは $range_{up}$より大きい場合、そのデータが MBR($Q\prime$) に含まれないので、18 行目で宣言した flag を 0 にして、次のデータを比較する。そうでない場合、そのデータが MBR($Q\prime$) の範囲内にあり CH(Q) にも含まれるので、出力となる商品集合 Q_m に挿入する。

4. 実　　験

提案手法の有効性を検証するため、DTM、前処理を加えた DTM の 2 つの手法で実験を行なった。全ての手法は C++で実装され、実験環境は macOS Sierra、プロセッサ 2.9GHz intel Core i5、メモリ 8G である。

4.1 実験データ

ユーザー好みベクトルの集合 $|U| = 100,000$、商品集合 $|P| = 100,000$、クエリ集合 $|Q| = 10$、k=10、クエリのレートの範囲 $q_{range} = 0$〜1 、商品のレイトの次元 D=3 を初期値として、$|Q|$、k、クエリのレートの範囲 q_{range}、次元 D を変量として実験を行なった。ユーザーの好みデータ U はすべて合成データを使用し、商品集合 P は合成データと実データ両方を使用してそれぞれ実験を行った。合成データは一様分布によって生成されたデータである。そして、実データは中国のレストラン口コミサイト dianping.com から取得したレストランの評価データであり、環境、サービス、価格、味わいという 4 次元の値が 0 〜 1 のレートを持つ。

Algorithm 1 クエリされた商品集合 Q の前処理

Require: Q
Ensure: Q_m

```
1:  Q′ ⇐ newArray, Q_m ⇐ newArray
2:  CH(Q) ⇐ makeCHofQ
3:  MBR(Q) =⇐ makeMBRofQ
4:  Q_low ⇐ MBR(Q).lowerLeftConner
5:  for i ∈ dimension(Q) do
6:      min ⇐ ∞
7:      for q ∈ Q do
8:          if q[i] − Q_low[i] < min then
9:              q′ ⇐ q
10:             Q′.push(q′)
11:         end if
12:     end for
13: end for
14: MBR(Q′) ⇐ makeMRBofQ′
15: range_low ⇐ MBR(Q′).lowerLeftConner
16: range_up ⇐ MBR(Q′).upperRightConner
17: for q ∈ CH(Q) do
18:     flag ⇐ 1
19:     for i ∈ dimension(Q′) do
20:         if q[i] > range_low[i] or q[i] < range_up[i] then
21:             flag ⇐ 0
22:             break
23:         end if
24:     end for
25:     if flag = 1 then
26:         Q_m.push(q)
27:     end if
28: end for
```

4.2 合成データの実験結果

合成データを用いて実験した結果は図 4 に示される。実験の初期値は 4.1 節で示されたように、ユーザー好み集合 $|U| = 100,000$、商品集合 $|P| = 100,000$、クエリ集合 $|Q| = 10$、k=10、クエリのレートの範囲 $q_{range} = 0$〜1 、商品のレートの次元 D=3 である。

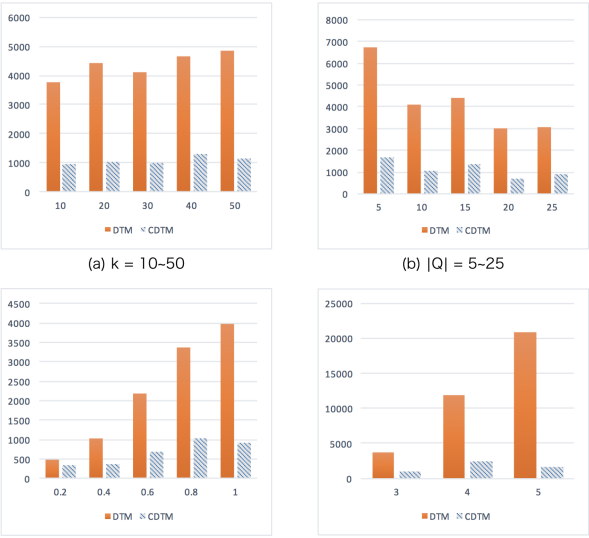
図 4.a では k を変量として、k = 10、20、30、40、50 で実験を行った結果である。いずれも従来の DTM より 3 倍以上の効果が示される。また、提案した前処理の効果が k に依存しないことが示される。

図 4.b では $|Q|$ を変量として、$|Q| = 5$、10、15、20、25 で実験を行った結果 である。こちらもすべて従来の DTM より 3 倍以上の効果が示されている。また、提案した前処理の効率がクエリ集合の数 $|Q|$ に依存しないことが示される。

図 4.c ではクエリのレートの範囲を 0〜0.2、0〜0.4、0〜0.6、0〜0.8、0〜1.0 で実験を行った結果である。クエリのレートの範囲が大きほど、従来の DTM と速度の差が大きいことが示

される。これは、従来の DTM ではクエリのレートの範囲が広がるほど BelowQ と AboveQ が小さくなり計算効率が悪くなるためである。本研究で提案した前処理は u によるランク付けする時の上限/下限から離れているデータを削除することで BelowQ/AboveQ が拡大され、高い計算効率が保たされる。

図 4.d では商品のレートの次元を変量として、D = 3、4、5 で行った実験結果である。次元数が増えるほど DTM の計算速度が著しく増加するが、前処理を加えることで計算速度が一定に保たされる。



(a) k = 10~50　(b) |Q| = 5~25

(c) qの範囲 = 0.2~1.0　(d) D = 3~5

図 4　合成データを用いた実験結果

4.3　実データの実験結果

中国のレストラン口コミサイト dianping.com から取得したデータを用いて実験を行った結果は図 5 に示される。実験の初期値は合成データの場合と同じ、ユーザー好み集合 $|U| = 100,000$、商品集合 $|P| = 100,000$、クエリ集合 $|Q| = 10$、k=10、クエリのレートの範囲 $q_{range} = 0$〜1 、商品のレートの次元 D=3 である。

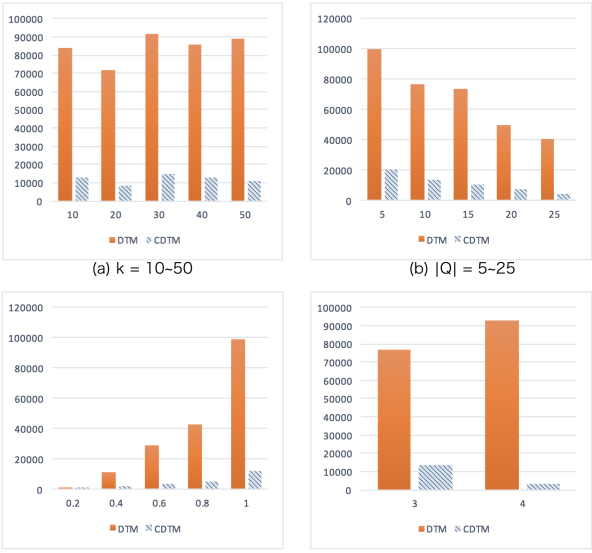
図 5.a では k を変量として、k = 10、20、30、40、50 で実験を行った結果である。合成データの場合と同じく、提案した前処理の効果が k に依存しないことが示される。計算速度は従来の DTM より 7 倍以上の速くなることが示される。

図 5.b では $|Q|$ を変量として、$|Q| = 5$、10、15、20、25 で実験を行った結果 である。こちらも合成データの場合と同じく、提案した前処理の効率がクエリ集合の数 $|Q|$ に依存しないことが示される。計算速度はいずれも従来の DTM より 5 倍以上の速くなることが示される。

図 5.c ではクエリのレートの範囲を 0〜0.2、0〜0.4、0〜0.6、0〜0.8、0〜1.0 で実験を行った結果である。クエリのレートの範囲が大きほど、従来の DTM と速度の差が大きくなり、合成データの場合と同じような効果が示されている。

図 5.d では商品のレートの次元を変量として、D = 3、4 で行った実験結果である。合成データでの実験結果と同じく、前処理を加えることで計算速度を一定に保つことができる。

全体的に、実データで行った実験は合成データで行った実験より効率が良かったことが示される。これは商品データの分布に影響された結果だと考えられる。合成データは一様分布によって生成されたため、データが空間上に散在するが、実データはレストランの環境、サービス、価格、味わいの評価であるため、データ空間上に集まっている。それによって AboveQ、BelowQ に含まれるデータ量の違いが大きく、以上の実験結果になったと考えられる。



(a) k = 10~50　(b) |Q| = 5~25

(c) qの範囲 = 0.2~1.0　(d) D = 3~4

図 5　実データを用いた実験結果

5.　お わ り に

本研究では、ARkR で提案された集約関数 Max/Min に対して効率的な検索手法を提案した。その手法ではクエリ集合 Q に対して前処理を加え、計算する必要のない $q \in Q$ を除外することで、先行研究で提案された TPM、DTM の手法で探索を除外できる領域を拡大し、計算コストを削減した。合成データとレストランの口コミサイトから取得したデータで実験を行った結果、提案したアルゴリズムは集約関数 Max/Min に対して従来の DTM 手法より 3 倍以上の効果が確認された。

文　　献

[1] Chang, Y.-C., Bergman, L.D., Castelli, V., Li, C.-S., Lo, M.-L., Smith, J.R.: The onion technique: Indexing for linear optimization queries. In: SIGMOD Conference, pp. 391402 (2000)

[2] Ilyas, I.F., Beskales, G., Soliman, M.A.: A survey of top-k query processing tech- niques in relational database systems. ACM Comput. Surv. 40(4), 11:111:58 (2008)

[3] Vlachou, A., Doulkeridis, C., Kotidis, Y., et al.: Reverse top-k queries. In: ICDE, pp. 365376 (2010)

[4] Vlachou, A., Doulkeridis, C. , Kotidis, Y., et al.: Monochromatic and bichromatic reverse top-k queries, pp. 12151229 (2011)

[5] Vlachou, A., Doulkeridis, C., Kotidis, Y., et al.: Branch-and-bound algorithm for reverse top-k queries. In: SIGMOD Conference, pp. 481492 (2013)

[6] Zhang, Z., Jin, C., Kang, Q.: Reverse k-ranks query.

PVLDB 7(10), 785796 (2014)
[7] Vlachou, A., Doulkeridis, C., et al.: Monitoring reverse top-k queries over mobile devices. In: MobiDE, pp. 1724 (2011)
[8] Dong Y., Chen H., Furuse K., Kitagawa H. (2016) Aggregate Reverse Rank Queries. In: Hartmann S., Ma H. (eds) Database and Expert Systems Applications. DEXA 2016. Lecture Notes in Computer Science, vol 9828. Springer, Cham