DEIM Forum 2016 D7-1

A Branch-and-Bound Method for Group Reverse Queries

Yuyang DONG†, Hanxiong CHEN†, Katazuka FURUSE†, and Hiroyuki KITAGAWA†

† Department of Computer Science, Information and systems, University of Tsukuba,
Ibaraki 305-8577, Japan
E-mail: †tou@dblab.is.tsukuba.ac.jp, ††{chx,furuse,kitagawa}@cs.tsukuba.ac.jp

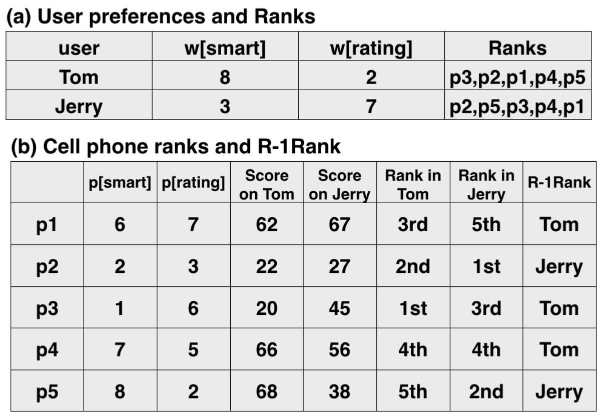
Abstract Recently, reverse ranks queries attracted significant interests in research. They applied for real-life applications such as market analysis and product placement. One of them, Reverse k-ranks queries returns user preferences that treat a given product be more favorite than others. Reverse k-ranks queries can help manufacturers find potential buyers even for an unpopular product. In marketing, manufacturers also willing to offer several products for sale as one combined product. Just like cable television industry often bundles channels and fast food industry combines separate food items into a complete meal. This kind of product bundling can help manufacturers extend market power even with imperfectly competitive products. Unfortunately, Reverse k-ranks queries and other reverse ranks queries can only for one product. To address this limitation, we propose a group reverse k-ranks queries to find k most potential user preference for a set of products. To solve group reverse k-ranks more efficiently, we propose a Tree pruning method (TPM) and a Branch-and-Bound pruning method (BPM). Theoretical analysis and experimental results show the efficacy of proposed methods.

Key words group reverse queries, branch-and-bound, product bundling

1. INTRODUCTION

Top-k queries and reverse k-ranks queries are two different kinds of view-model. Top-k queries is a user-view model that support to consumers by obtaining the best k products for a user preference. On the other hand, reverse k-ranks [12] queries charge to manufactures to discover the potential consumers by retrieving the most appropriate user preference than others. So it is a manufacture view mode. From the perspective of manufacturers, reverse k-ranks can be a tool for identifying the customers and estimate the marketing of products.

Figure 1 shows the example of reverse k-ranks query when $k = 1$. There are 5 different cell phones ($p_1 \sim p_5$) and the scores of "smart" and the "rating", are recorded in table (a). Two users preference (Tom and Jerry) showed in table (b); these preferences consist of the weights for each attribute. The scores of a cell phone based on a user preference is the result of the inner product between the cell phone attributes vector and user preference vector. Without loss of generality, in this paper, minimum scores will be preferable. The result of Reverse 1-ranks queries showed at the last cells in Table (b). For example, Tom thinks cell phone p_1 is the 3rd best phone, Jerry thinks it is the 5th. To manufacture, Tom has more possibility than Jerry to buy cell phone p_1, so reverse 1-rank query returns Tom as the result.



(a) User preferences and Ranks

user	w[smart]	w[rating]	Ranks
Tom	8	2	p3,p2,p1,p4,p5
Jerry	3	7	p2,p5,p3,p4,p1

(b) Cell phone ranks and R-1Rank

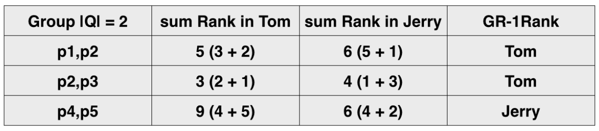
	p[smart]	p[rating]	Score on Tom	Score on Jerry	Rank in Tom	Rank in Jerry	R-1Rank
p1	6	7	62	67	3rd	5th	Tom
p2	2	3	22	27	2nd	1st	Jerry
p3	1	6	20	45	1st	3rd	Tom
p4	7	5	66	56	4th	4th	Tom
p5	8	2	68	38	5th	2nd	Jerry

Figure 1: The example of reverse 1-rank queries.

1.1 motivation

In marketing, manufacturers also do sell with Product Bundling. Product Bundling is offering several products for sale as one combined product. It is a common feature in many imperfectly competitive product markets. For examples, Microsoft Co., Ltd include a word processor, spreadsheet, presentation program and other useful software into a single Office suite. The cable television industry often bundles various channels into a single tier to expand channel market. Manufacturers of Video games also willing to group a popular game with other games with the same theme, they can obtain more benefits by selling them together.

As the Product Bundling is an important approach for



Group lQl = 2	sum Rank in Tom	sum Rank in Jerry	GR-1Rank
p1,p2	5 (3 + 2)	6 (5 + 1)	Tom
p2,p3	3 (2 + 1)	4 (1 + 3)	Tom
p4,p5	9 (4 + 5)	6 (4 + 2)	Jerry

Figure 2: The example of group reverse 1-rank queries.

sale, it is significant to help manufacturers find the most probable buyers for their bundled products. Unfortunately, the reverse k-ranks query and other kinds of reverse ranking queries are all designed for just one product. To solve this limitation, we propose a new query definition that finds k customers with smallest group rank values, where the rank of the group defined as the sum of each product's rank. We name this query as group reverse k-rank queries (GR-k in abbreviation).

Figure 2 shows the example of GR-1 queries. There are 3 groups of Bundling Product sale, $\{p_1, p_2\}$, $\{p_2, p_3\}$ and $\{p_4, p_5\}$. The group rank of $\{p_1, p_2\}$ is 5 by Tom's preference and is 6 by Jerry's. So GR-1 query return Tom as the result.

1. 2 Contribution

The contribution of this paper is:

- To address these "one product" limitation from reverse k-ranks queries. We define a new ranking query named group reverse k-ranks query (GR-k) that return best k user preferences to match a set of products. The GR-k can support to Product Bundling.
- We propose the solutions to process GR-k query efficiently. They are tree pruning method (TPM) and branch-and-bound pruning method (BPM).
- We employ the experiments both on factual data and synthetic data. The experimental results evaluate the efficiency of the proposed methods.

2. RELATED WORK

User-view model queries, top-k queries. The Onion technique [1] precomputes and stores the convex hulls of data points in layers like an onion. The evaluation of a linear top-k query is accomplished by starting from the outmost and processing these layers inward.

Reverse top-k queries [5, 6] have been proposed for evaluating the impact of a potential product in the market, based on preferences of users that treat this product as one of their top-k products. There are also various applications of reverse top-k queries, [9] identifying the most influential products, [8] monitoring the popularity of locations based on user mobility. [7] propose a branch-and-bound algorithm for reverse top-k queries.

However, in order to answer reverse query for some less popular objects, [12] proposed reverse k-ranks queries, which finds for a given object the top-k user preferences whose rank for the object is highest among all users.

Some other related researches on reverse query are listed in the following. The essential difference is that they treat one data set, while in reverse rank queries, there are two data sets. Given a data point and aim at finding the queries that have this data point in their result set. Contrast with the nearest neighbour search, [3] proposed a reverse nearest neighbour (RNN) queries. Opposite to RNN, [11] proposed reverse furthest neighbour (RFN) queries to find the points who deem query point as their furthest neighbour. [10] classifies RKNN (reverse k nearest neighbour) into bichromatic and monochromatic queries. This make RKNN be similar with reverse rank queries that obtaining users that prefer given product as favorite, but they are different queries. RKNN evaluates relative L_p distance in one euclid space with two points. However, reverse rank queries focus on the absolute ranking among all objects, and scores are found by doing inner product (weighted sum) with two different vectors, user preference and object, and they from different data space. The reverse skyline query uses the advantages of products to find the potential customers based on the dominance of the competitors products [2, 4].

3. PROBLEM STATMENT

There are a data set P and a data set W all in d dimension. Each product $p \in P$ is a vector which contains d non-negative scoring attribute. p can represented as $p = (p[1], p[2], ..., p[d])$, and $p[i]$ is an attribute value of i'th dimension. The user preference $w \in W$ is also a vector, and $w[i]$ corresponds to $p[i]$ so that evaluate the i'th attribute of p, where $w[i] \geqq 0$ and $\sum_{i=1}^{d} = 1$.

The score of p on w is defined by doing inner product as $f_w(p) = \sum_{i=1}^{d} w[i] \cdot p[i]$. Without loss of generality, we consider smaller score values are preferable that same with related research.

The definitions of reverse k-ranks queries [12] are as follow.

[Definition 1] (reverse k-ranks query). Given a point set P, a weighting vector set W, a positive integer k, and a query point q. R-kRanks query returns a set S, $S \subset P$, $|S| = k$, such that $\forall w_i \in S, \forall w_j \in (W - S)$, $rank(w_i, q) \leqq rank(w_j, q)$ holds. Where $rank(w, q) = |R|$, where $|R|$ is the cardinality of R, a subset of P. $\forall p_i \in R$, we have $f(w, p_i) < f(w, q)$ and $\forall p_j \in (P - S)$, $f(w, p_j) \geqq f(w, q)$.

The group rank is defined by the sum of each rank of q, so

[Definition 2] ($gRank(w, Q)$) Given a point set P, a weighting vector w, and a query points set Q, the group rank of Q for w is $gRank(w, Q) = \sum_{i=1}^{|Q|} rank(w, q_i)$, where

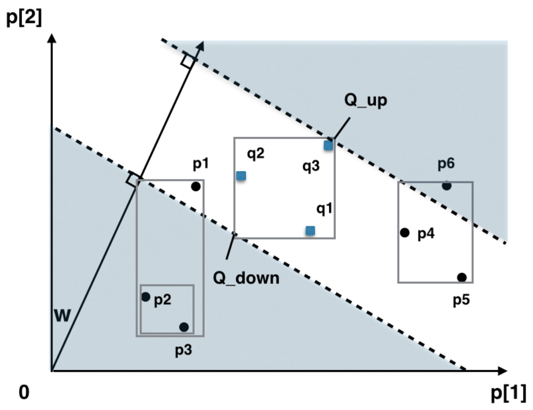


Figure 3: The filtering part (blue) of TPM algorithm with 2-dimension data.

$q_i \in Q$.

The definition of group reverse k-ranks queries is:

[Definition 3] (group reverse k-ranks query). Given a point set P, a weighting vector set W, a positive integer k, and a query point set Q. GR-kRanks query returns a set S, $S \subset P$, $|S| = k$, such that $\forall w_i \in S, \forall w_j \in (W - S)$, $gRank(w_i, Q) \leqq gRank(w_j, Q)$ holds.

4. Tree Pruning Method

Algorithm 1 TraRtreeP

```
Require: $P, w, Q, minRank$
Ensure: include: rnk; discard: -1;
 1: $rnk \Leftarrow 0$
 2: $heapP.enqueue(RtreeP.Root())$
 3: while $heapP.isNotEmpty()$ do
 4:     $e \Leftarrow heapP.dequeue()$
 5:     for each $e_i \in e$ do
 6:         if $f(w, e_i.up) < f(w, Q.down)$ then
 7:             $rnk \Leftarrow rnk + e_i.size() * |Q|$
 8:             if $rnk \geqq minRank$ then
 9:                 return -1
10:         else if $f(w, e_i.down) > f(w, Q.up)$ then
11:             coutinue
12:         else
13:             if $e_i$ is a leaf node then
14:                 for each $q \in Q$ do
15:                     if $f(w, q) > f(w, e_i)$ then
16:                         $rnk + +$
17:                         if $rnk \geqq minRank$ then
18:                             return -1
19:             else
20:                 $heapP.enqueue(e_i)$
21: return rnk
```

Processing group reverse k-ranks queries with the big data set is a challenge since the naive algorithm needs to evaluate every pair of product and user preference. It will cost vast amounts of computation with a large number of data set of products and user preferences. To boost efficiency, we index the products data set with R-tree for grouping similar products and use upper and lower bounders of MBRs (Minimum Bounding Rectangles) to save computing. We call it tree pruning method (**TPM**). The Geometry view of TPM algorithm is showed in Figure 3. The blue shadow part, form by dash line and space border, below $Q.down$ or above $Q.up$ is the part that TPM algorithm can filter.

Base on the feature of MBRs in R-tree, we can use the two facts below to pruning unnecessary MBRs entirely.

Fact 1. For a query point set Q, a weighting vector w, a MBR e of R-tree. If $f(w, e.up) < f(w, Q.down)$, $\forall p \in e, \forall q \in Q$, it holds $f(w, q) < f(w, p)$.

Fact 2. For a query point set Q, a weighting vector w, a MBR e of R-tree. If $f(w, e.down) < f(w, Q.up)$, $\forall p \in e, \forall q \in Q$, it holds $f(w, q) > f(w, p)$.

Algorithm 2 Tree Pruning Method (**TPM**)

```
Require: $P, W, Q$
Ensure: result set $heap$
 1: $heap \Leftarrow \emptyset$
 2: for each $w \in \{$first $k$ element in $W\}$ do
 3:     $heap.insert(w, gRank(w, Q))$
 4: $minRank \Leftarrow heap$'s last rank.
 5: for each $w \in W-$ {first k element in $W$} do
 6:     $rnk \Leftarrow TraRtreeP(P, w, Q, minRank)$
 7:     if $rnk \neq -1$ then
 8:         $heap.insert(w, rnk)$
 9:         $minRank \Leftarrow heap$'s last rank.
10: return heap
```

4.1 TraRtreeP algorithm

When given data set P, a weighting vector w, a query point set Q and a positive integer k. The $TraRtreeP$ algorithm can return whether the group rank of Q is smaller than the given $minRank$ or not. As the algorithm 1 shows, $TraRtreeP$ algorithm use R-tree to prune similar points in a group (MBR). In this algorithm, we use counter rnk to counter the group rank of Q (Line 1). Then we recursively check the MBRs in products R-tree from the root (Line 2). If the upper bounder score of current entry e_i smaller than query set Q's lower bound score, the counter rnk increased by $e_i.size() * Q$ (Line 6-7). When rnk get greater than $minRank$, it means this Q should not rank in $minRank$ on w and the algorithm return -1 to terminate (Line 8-9). If lower bound score of current entry greater than Q's upper bound score, we don't need to check any child in this entry and just skip this entry (Line 10-11). Other situation, when checking a leaf node of entries, we compute the score of each $q \in Q$ on w and increase rnk (Line 13-18). If the algorithm does not return in the recursion, we return rnk as the rank

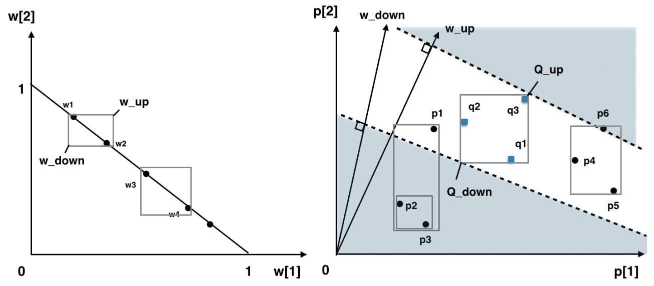


Figure 4: The filtering part (blue) of BPM algorithm with 2-dimension data.

of this Q based on the weighting vector w.

4.2 TPM algorithm.

In TPM algorithm, we first initialize a *heap* with first k weighting vectors and their ranks (Line 1-3). Then for others weighting vectors, we call Algorithm 1 to check the group rank of the query set Q (Line 6). If the current w can make Q's rank better than the last rank in *heap*, this w will be inserted into *heap* with its rank. The *heap* will auto-update itself by removing the last element and insert a new w and a group rank, and then the heap sort itself by the order of rank (Line 8). After update *heap*, *minRank* also updated by the last rank in *heap* (Line 9). At last, the algorithm returns *heap* as the result of group reverse k-ranks query.

5. Branch and Bound Pruning Method

TPM use R-tree to management the similar products and avoid computing by MBRs. However, there is a limitation that TPM evaluated each user preference one by one, and it will reduce the efficiency when encountering a large weight vectors data set. This limitation inspires us to remove redundant computing by grouping similar weighting vectors. We propose the branch-and-bound pruning method (**BPM**). The Figure 4 shows the filtering property of BPM algorithm. We denote the R-tree for products and weighting vectors as *RtreeP* and *RtreeW* respectively.

Based on the feature of MBRs in *RtreeP* and *RtreeW*, there also have two facts in the following to pruning unnecessary MBRs entirely.

Fact 3. For a query point set Q, a MBR e_p of R-tree$P$ and a MBR W_e of *RtreeW*. If $f(W_e.up, e_p.up) < f(W_e.down, Q.down)$, then for $\forall p \in e_p, \forall w \in W_e, \forall q \in Q$, it holds $f(w,q) < f(w,p)$.

Fact 4. For a query point set Q, a MBR e of R-tree$P$ and a MBR W_e of *RtreeW*. If $f(W_e.down, e_p.down) < f(W_e.up, Q.up)$, then for $\forall p \in e_p, \forall w \in W_e, \forall q \in Q$, it holds $f(w,q) > f(w,p)$.

5.1 Traverse RtreeP with W_e

For a MBR of weighting vectors from R-treeW, denoted as W_e, Algorithm 3 helps to check these weighting vectors with Q and *minRank*. The algorithm returns 1 if all $w \in W_e$ make Q rank in *minRank*, and returns -1 if none of $w \in W_e$ make Q rank in *minRank*. When some of W_e may let Q rank better than *minRank*, the algorithm returns 0.

The difference with TPM algorithm stated in Section 4., BPM use two R-tree to index weighting vectors and products. Based on the methodology of branch-and-bound, BPM can prune both weighting vectors and products.

5.2 BPM algorithm

Algorithm 3 TraRtreeP++

```
Require: $P, W_e, Q, minRank$
Ensure: include: 1; discard: -1; 0: uncertain
 1: $rnk \Leftarrow 0$, $rnkUp \Leftarrow 0$
 2: $heapP.enqueue(RtreeP.root())$
 3: while $heapP.isNotEmpty()$ do
 4:   $e \Leftarrow heapP.dequeue()$
 5:   for each $e_i \subset e$ do
 6:     if $f(W_e.up, Q.up) > f(W_e.down, e_i.down)$ then
 7:       if $f(W_e.up, e_i.up) < f(W_e.down, Q.down)$ then
 8:         $rnk \Leftarrow rnk + e_i.size() * |Q|$
 9:       else if $f(W_e.down, e_i.down) > f(W_e.up, Q.up)$ then
10:         coutinue
11:       else
12:         if $e_i$ is a leaf node of $RtreeP$ then
13:           for each $q \in Q$ do
14:             if $f(W_e.down, q) > f(W_e.up, e_i)$ then
15:               $rnk++$
16:               if $rnk \geqq minRank$ then
17:                 return -1
18:             if $f(W_e.up, q) > f(W_e.up, e_i)$ then
19:               $rnkUp++$
20:         else
21:           $heapP.enqueue(e_i)$
22: if $rnk + rnkUp \leqq minRank$ then
23:   return 1
24: else
25:   return 0
```

In Algorithm 4, we start from the root of *RtreeW* and call Algorithm 3 to check the W_e (Line 14). If the $flag$ is 0, we add all sub MBRs into *heapW* for next loop (Line 9-10). If the $flag$ is 1, it means every weighting vector in W_e make Q rank better than *minRank*, so we call Algorithm 1 to compute the rank of weighting vectors in W_e and update the *heap* and *minRank* (Line 20-22). When the leaf node of single weighting vector need to check, we also use Algorithm 1 as same as the way in TPM (Line 8-12). At the end of the algorithm, we return *heap* as the result of group reverse k-ranks query.

6. EXPERIMENT

We present the experimental evaluation of the TPM and

Algorithm 4 branch-and-bound pruning method (**BPM**)

```
Require: $P, W, Q$
Ensure: result set $heap$
1: $heap \Leftarrow \emptyset$
2: for each $w \in$ {first $k$ element in $W$} do
3:     $heap.insert(w, gRank(w, Q))$
4: $minRank \Leftarrow heap$'s last rank.
5: $heapW.enqueue(RtreeW.root())$
6: while $heapW.isNotEmpty()$ do
7:     $W_e \Leftarrow heapW.dequeue()$
8:     if $W_e$ is a leaf node then
9:         $rnk \Leftarrow TraRtreeP(P, w, Q, minRank)$
10:        if $rnk \neq -1$ then
11:            $heap.insert(w, rnk)$
12:            $minRank \Leftarrow heap$'s last rank.
13:    else
14:        $flag \Leftarrow TraRtreeP++(P, W_e, Q, minRank)$
15:        if $flag == 0$ then
16:            $heapW.enqueue$(all subMBR $\subset W_e$)
17:        else
18:            if $flag == 1$ then
19:                for each $w \in W_e$ do
20:                    $rnk \Leftarrow TraRtreeP(P, w, Q, minRank)$
21:                    $heap.insert(w, rnk)$
22:                    $minRank \Leftarrow heap$'s last rank.
23: return $heap$
```

Parameter	Values
Data dimensionality d	$2 \sim 5$, **3**
Distribution of data set P	**UN**,CL,RE
Distribution of data set W	**UN**,CL
Data set cardinality of $\|W\|$ and $\|P\|$	100K
Experiment times	1000
Number of clusters	$\sqrt[3]{\|P\|}$, $\sqrt[3]{\|W\|}$
Variance σ_W^2,σ_P^2	0.1^2

Table 1: Experimental parameters and default values(in bold) .

BPM algorithms for aggregate reverse k-rank. All algorithms are implemented in C++ and the experiments run on a Mac with 2.6 GHz Intel Core i7, 16GB RAM, and 500GB flash storage, OS X Yosemite.

6. 1 Experiment setup

Data set P. We employed both synthetic data and real data for products P. Synthetic data set are uniform (UN) and clustered (CL), whose attribute value range is $[0, 1)$. In UN, all attributes are generated independently following a uniform distribution. CL is generated by randomly selecting M centroids ($M = \sqrt[3]{|P|}$) which following a uniform distribution. Each coordinate is generated following the normal distribution with variance $\sigma = 0.1$ and mean equal to the corresponding coordinate of the centroid. The real data, named HOUSE, contains 201760 6-dimensional tuples, representing American family annual payment on gas, electricity, water, heating, insurance, and property tax in 2013. The dataset HOUSE also used into related search [5, 7].

Data set W. For data set W, we also have UN, CL data set that is in a same generating way with data set P.

Parameters. Parameters are shown in Table 1 where the default values are $d = 3$, $|P|$=100K, $|W|$=100K, $k = 10$, $|Q|$ = 5, and both P and W are UN data.

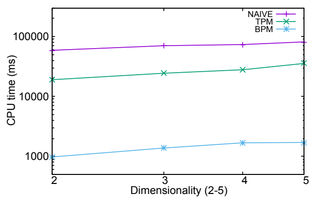
Metrics. Our metrics is the query execution time required by each algorithm. We do each experiment over 1000 times and present the average value. The query point set Q is randomly selected from P. To insight the filtering effect of TPM and BPM, we also observe the computing time of real score for a point in the leaf node of R-tree.

6. 2 Experiment result

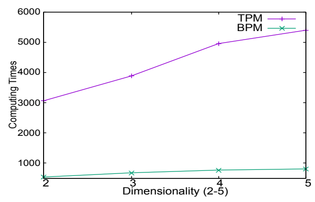
Un data. Figure 5 shows the experimental results on uniform distribution data sets on varying dimension d (2-5). Figure 5a shows the CPU time of implemented algorithms, NAIVE, TPM and BPM. According to the result, we can see that TPM is much faster than NAIVE algorithm.Notice that y-axis is in the log. BPM is more than ten times faster than other algorithms and is stable even in 5 dimensions. Figure 5b shows computing times when the iterator access the leaf node of R-tree. BPM algorithm has less computation in a processing means that it prunes more unnecessary points and weighting vectors than TPM.

CL data. Figure 6 shows the experimental results on Cluster data sets on varying dimension d (2-5). Figure 6a shows the CPU time of implemented algorithms while Figure 6b shows computing times. Notice that all tree-base methods (TPM and BPM) take less time finish querying than UN data because Cluster data is easy to index by R-tree. The experimental result is same with UN data, TPM is the best algorithms and has the least computation.

House data. Figure 7 shows the performance of real data

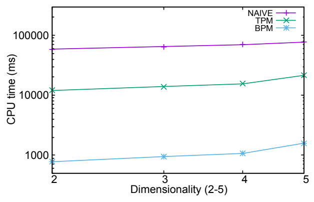


(a) Processing time.

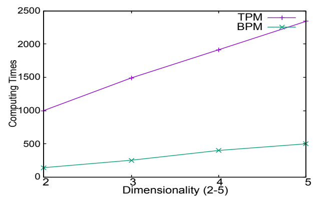


(b) Computing times.

Figure 5: Experimental result on UN data, $|P| = 100K$, $|W| = 100K$, $|Q| = 5$, $k = 10$, $d = 2 \sim 5$.

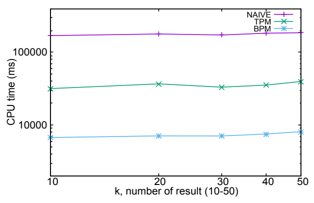


(a) Processing time .

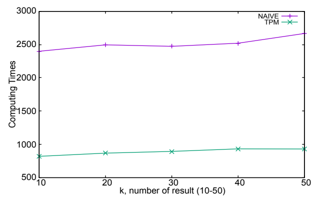


(b) Computing times.

Figure 6: Experimental result on CL data, $|P| = 100K$, $|W| = 100K$, $|Q| = 5$, $k = 10$, $d = 2 \sim 5$.



(a) Processing time.



(b) Computing times.

Figure 7: Experimental result on HOUSE data, $|P| = 100K$, $|W| = 100K$, $|Q| = 5$, $k = 10$, $d = 2 \sim 5$.

set HOUSE. Figure 7a and 7b is the result of CPU cost and computing time respectively. Clearly, both TPM and BPM faster than NAIVE and TPM is the best one. The BPM is faster than TPM and has less computation amount than TPM.

7. CONCLUSION

Reverse rank queries have become important tools in marketing analyzing. However, the related researches of reverse rank queries are all for one product. We propose group reverse k-ranks queries to apply reverse rank queries into the situation of several products like the Product Bundling. We devise two methods, TPM and BPM, to answer group reverse k-ranks queries efficiently. Tree Pruning Method (TPM) is a tree-based pruning method prune unnecessary products with the help of R-tree. Branch-and-Bound Pruning Method (BPM) uses two R-tree to manage Products and User preferences, the unnecessary Products and User preferences can be pruned at the same time based on the methodology of Branch-and-Bound. We do the comparison experiment on both synthetic data and real data, the experimental verify the efficiency of our proposed methods.

As future work, we have two objects. The first one is to consider another evaluating function. e.g., Evaluating a group of products by the harmonic average of each one's rank instead of the sum function in this paper. The second one consider the approximate solutions for group reverse k-ranks queries.

References

[1] Chang, Y.-C., Bergman, L. D., Castelli, V., Li, C.-S., Lo, M.-L., and Smith, J. R. The onion technique: Indexing for linear optimization queries. In *SIGMOD Conference* (2000), pp. 391–402.

[2] Dellis, E., and Seeger, B. Efficient computation of reverse skyline queries. In *Proceedings of the 33rd International Conference on Very Large Data Bases, University of Vienna, Austria, September 23-27, 2007* (2007), pp. 291–302.

[3] Korn, F., and Muthukrishnan, S. Influence sets based on reverse nearest neighbor queries. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, USA.* (2000), pp. 201–212.

[4] Lian, X., and Chen, L. Monochromatic and bichromatic reverse skyline search over uncertain databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008* (2008), pp. 213–226.

[5] Vlachou, A., Doulkeridis, C., and Kotidis, Y. Reverse top-k queries. In *ICDE* (2010), pp. 365–376.

[6] Vlachou, A., Doulkeridis, C., and Kotidis, Y. Monochromatic and bichromatic reverse top-k queries. pp. 1215–1229.

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

[8] Vlachou, A., Doulkeridis, C. Monitoring reverse top-k queries over mobile devices. In *MobiDE* (2011), pp. 17–24.

[9] Vlachou, A., Doulkeridis, C., and Kotidis, Y. Identifying the most influential data objects with reverse top-k queries. pp. 364–372.

[10] Yang, S., Cheema, M. A., Lin, X., and Zhang, Y. SLICE: reviving regions-based pruning for reverse k nearest neighbors queries. In *IEEE 30th International Conference on Data Engineering, Chicago, ICDE 2014, IL, USA, March 31 - April 4, 2014* (2014), pp. 760–771.

[11] Yao, B., Li, F., and Kumar, P. Reverse furthest neighbors in spatial databases. In *Proceedings of the 25th International Conference on Data Engineering, ICDE 2009, March 29 2009 - April 2 2009, Shanghai, China* (2009), pp. 664–675.

[12] Zhang, Z., Jin, C., and Kang, Q. Reverse k-ranks query. *PVLDB 7*, 10 (2014), 785–796.