DEIM Forum 2017 G1-1

A High-dimensional Solution for Aggregate Reverse Rank Query

Yuyang DONG[†], Song WANG[†], 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, ††wangsongpower@163.com, †††{chx,furuse,kitagawa}@cs.tsukuba.ac.jp

Abstract In user-product model, there are two types of datasets: Products and users. The Top-rank query in user-product model can help users to find matching products in their top-rank. On the other hand, the aggregate reverse rank query (ARR) can find potential users for multiple products (product bundling). ARR is an essential tool in marketing analysis that helps manufacturers identify the placement of their product bundling. Unfortunately, the previous solution for the ARR query is a tree-based method. Due to the curse of dimensionality, the tree-based method makes ARR query don't have good performance in high-dimensional data. To address this limitation, we use a high-dimensional technical named Grid-index, and propose a Grid-index aggregate (GIA) solution to deal with the ARR query in high-dimensional data. Our experimental results show that the GIA has better performance than the existing tree-based algorithms.

Key words Aggregate Reverse Rank Queries, Grid-index, High dimensional index

1. INTRODUCTION

Top-k queries is a user-view model that obtain the best k products for a user preference. On the other hand, aggregate reverse rank query (ARR) [3] is a manufacturer-view model that discovers the potential consumers by retrieving the most appropriate user preferences for multiple products. ARR can help manufacturers with the market analysis with their product bundling.

Figure 1 shows the example of aggregate reverse ranks query when $k = 1$. There are 5 different books ($p_1 \sim p_5$) with two attributes "price" and the "rating" in Table (b). Two users preferences (Tom and Jerry) are shown in Table (a); these preferences consist of the weights for each attribute of a book. The score of a book based on a user preference is the result of the inner product between the book attributes vector and user preference vector. All scores are ranked with scores and we think the minimum scores will be preferable. The ranking results of books by users are shown in the last cells Table (a). Table (c) shows that the bookshop offers two kinds of product bundling, to bundle $\{p_1, p_2\}$ and $\{p_4, p_5\}$. ARR evaluates aggregate rank with the sum of each book, so the bundle of $\{p_1, p_2\}$ ranked as $3 + 2 = 5$ based on Tom's preference. For $\{p_1, p_2\}$, Tom thinks the aggregate rank is 5, and Jerry thinks it is the 6th. ARR-1 returns Tom as a result since he has more possibility than Jerry to buy the bundling.

(a) User preferences and Ranks

	w[price]	w[rating]	Ranking
Tom	0.8	0.2	p_3,p_2,p_1,p_4,p_5
Jerry	0.3	0.7	p_2,p_5,p_3,p_4,p_1

(b) Books

	p[price]	p[rating]	Rank on Tom	Rank on Jerry
p_1	6	7	3rd	5th
p_2	2	3	2nd	1st
p_3	1	6	1st	3rd
p_4	7	5	4th	4th
p_5	8	2	5th	2nd

(c) Bundled books and ARR-1

	ARank on Tom	ARank on Jerry	AR-1Rank
p_1,p_2	5 (3+2)	6 (5+1)	Tom
p_4,p_5	9 (4+5)	6 (4+2)	Jerry

Figure 1: Users data, books data and ARR-1 results.

1.1 Motivation

The state-of-the-art solution for ARR is the *double tree method* (DTM). It is a tree-based methodology, which uses an R-tree to indexes and prunes the data through a branch-and-bound method. However, the R-tree or any other spatial index has a shortcoming: When processing high-dimensional data sets, the performance declines to close to NAIVE.

We do an experiment to observe the consuming time of processing ARR. Figure 2 shows the comparison of performance

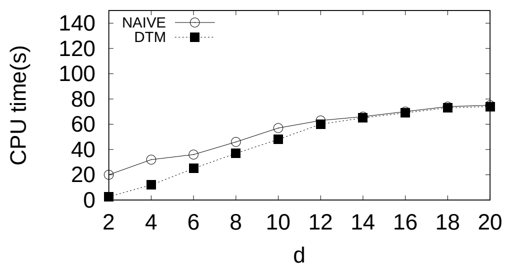

Figure 2: DTM vs NAIVE.

between tree-based algorithm (DTM) and the NAIVE. According to the results, DTM may not outperform the NAIVE method when processing ARR in high dimensions.

However, both the product's attributes and the user's preferences are likely to be high-dimensional. For example, cell phones consumers care about many features, such as price, processor, storage, size, battery life, camera, etc. Therefore, processing ARR with a high-dimensional data set is an important requirement.

2. RELATED WORK

For 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 [6, 7, 9, 10] 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, 10]. However, to answer the reverse query for some less popular objects, [13] 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. The most relevant work is our previous work named aggregate reverse rank query (ARR) [3]. It finds the top-k users for multiple query products to deal with the application of product bundling.

Some other related researches on the 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 neighbor search, [4] proposed a reverse nearest neighbor (RNN) queries. Opposite to RNN, [12] proposed reverse furthest neighbor (RFN) queries to find the points who deem query point as their furthest neighbor. [11] classifies RKNN (reverse k nearest neighbor) into bichromatic and monochromatic queries. RKNN may look similar with reverse rank queries that obtaining users that prefer a given product as favorite, but they are completely different. 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 inner product with two different vectors from different data space. The reverse skyline query uses the advantages of products to find the potential customers based on the dominance of the competitor's products [2, 5].

3. PRELIMINARIES

3.1 Definitions

We first introduce the assumption of the product database, preference database and the score function, which is same with the related research [3, 6, 8, 13]. Assume that there is a product data set P and a preference data set W. Each $p \in P$ is a d-dimensional vector that contains d non-negative scoring attributes. p can also be represented as a point $p = (p[1], p[2], ..., p[d])$, where $p[i]$ is the ith attribute value of p. i.e., the value on ith dimension of p. The preference $w \in W$ is a d-dimensional weighting vector, and $w[i]$ is a non-negative weight that evaluates $p[i]$, where $\sum_{i=1}^{d} w[i] = 1$. The score is defined as the inner product of p and w, which is expressed by $f(w, p) = \sum_{i=1}^{d} w[i] \cdot p[i]$. A query set Q, is set of points in the space of product P (but it is not necessary that $Q \subset P$). All values are normalized to the range (0,1].

The rank of a q on a specific w is defined as the number of points which have smaller scores than q:

[Definition 1] ($rank(w, q)$). Given a product dataset P, a preference w, and a query q, the rank of q according to w is $rank(w, q) = |S|$, where $S \subseteqq P$ and $\forall p_i \in S, f(w, p_i) < f(w, q) \land \forall p_j \in (P - S), f(w, p_j) \geqq f(w, q)$.

To evaluate the rank of a query set Q, related work [3] gave a definition that the aggregate rank of Q is the sum of each rank of $q \in Q$:

[Definition 1] (rank) ($ARank(w, Q)$). The aggregate rank of Q according to a w is$ARank(w, Q) = \sum_{q_i \in Q} rank(w, q_i)$.

The definition of *aggregate reverse rank query* [3] is:

[Definition 2] (aggregate reverse rank query, ARR). Given datasets P and W, positive integer k, and query product set Q, ARR query returns the set S, $S \subseteqq W$, $|S| = k$, such that $\forall w_i \in S, \forall w_j \in (W - S)$, $ARank(w_i, Q) \leqq ARank(w_j, Q)$ holds.

3.2 Tree-based Method for ARR

The state-of-the-art solution for ARR is double tree method (DTM) [3]. DTM uses two R-trees to index data P and data W respectively, then filters unnecessary data with the branch-and-bound methodology. The node of R-tree (minimum bounding rectangle, MBR) can help to compare multiple p's and w's at one time by using the MBRs' bounds. As shown in Figure 3, the left is the space of data W, and the right is the space of data P. An R-tree node e_w groups w_1, w_2, w_3 and it's upper bound $e_w.up$ and lower bound $e_w.down$ form the search space (gray area) with Q. DTM only need do computing of the data in the sandwiched space, the nodes like e_2, e_3 and e_5 will be pruned directly.

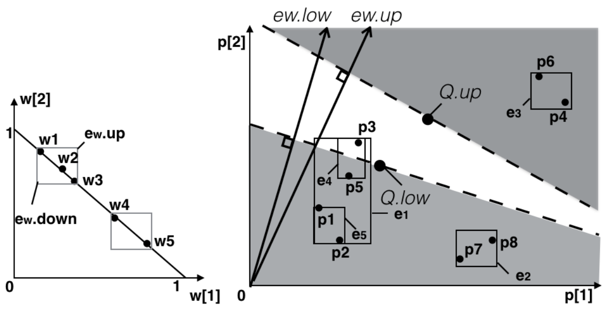

Figure 3: Tree-based method of ARR query.

4. Why tree-based method performed *ARR* not good with high-dimensional data?

In this section, we demonstrate why the tree-based method has poor performance when processing ARR with high-dimensional data. There are two reasons: Overlapped nodes and small filter space.

4.1 Overlapped Nodes

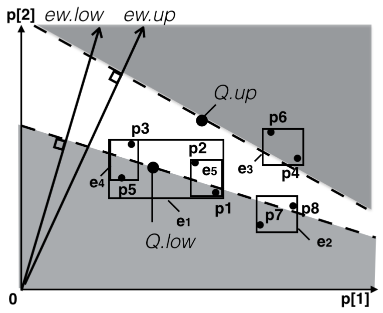

Figure 4: Overlapped Nodes.

It is well known that R-tree groups similar points as a node with their MBR, however, the nodes will be overlapped with each other when indexing high-dimensional data. Due to the high-dimensional case cannot be plotted, we use a 2-dimensional example to demonstrate the overlapped nodes. In Figure 4, an R-tree indexes data points into five nodes e_1 to e_5. Unfortunately, all nodes all overlapped each other and intersected with the borderlines of search space (dashed line). This time, DTM can prune nothing while doing a traversal of R-tree and have to compute all data points in the leaf node. The above situation always happens in high-dimensional space.

4.2 Small Filter Space

The reason of overlapped nodes comes from the tree-structure itself. However, another reason for the bad performance is because of the filtering property declines in high-dimensional space.

As Figure 4 shows, the shape of the gray area can be a prism or a tetra. The filtering property can be estimated by measuring the volume of :

$$Vol = Vol_{TetraX} \cdot Vol_{PrismX} + Vol_{TetraY} \cdot Vol_{PrismY} \quad (1)$$

To give an analytical result, we assume that the upper filtering volume are equal to lower filtering volume ($Vol_{TetraX} = Vol_{TetraY}$), hence:

$$Vol = 2 \cdot Vol_{Tetra} \cdot Vol_{Prism} \quad (2)$$

the volume of g-dimensional hyper-tetra is:

$$Vol_{Tetra} = \frac{1}{g!}(\prod_{i=1}^{g} x_i) = \frac{1}{g!}(1-\gamma)^g \quad (3)$$

the volume of the hyper-prism is:

$$S_i = \frac{1}{2}(x_i + x_i') \cdot H \leqq (\frac{1-\gamma}{2}) \leqq \frac{1}{2} \quad (4)$$

Recall that the length of the range is 1, so a 3 dimensional prism in the figure, the volume is:

$$Vol_{Prism3d} = \frac{1}{3}(S_1 + S_2 + \sqrt{S_1 S_2}) \cdot H \leqq \frac{1}{2} \quad (5)$$

This result holds for higher dimensional prisms. The maximum volume of the filtered space is as following, where $g = d/2$ and means there are same numbers of tetra or prism.

$$Vol_{max} = 2 \cdot \frac{1}{g!}(1-\gamma)^g \cdot \frac{1}{2} = \frac{1}{g!}(1-\gamma)^g \quad (6)$$

According to above Equation 6, the volume will be very small in high-dimensional cases, which proved that the filterable space also becomes small.

5. Proposed Grid-Index

As we know that the most cost part in processing ARR the linear scan is the computing of the scores (inner products) for comparison. A d dimensional inner product needs d times multiplications and d times additions, and the multiplications cost much more than additions. To enhance the efficiency of the linear scan, we introduce the concept of Grid-index which can help to avoid most multiplications.

5.1 Build Grid-index

We first partition the value range of data p and w, notice that all attributes of $p \in W$ should belong to the same range, so do all $w \in W$. In the example of Figure 5, we partition the value range into 4 equal intervals. For the given

$p = (0.62, 0.15, 0.73)$, $p[1] = 0.62$ falls into the third partition $[0.5, 0.75]$. $p[2] = 0.15$, falls into the first partition, and $p[3]$ is in the third partition. We will store the partition numbers as an approximate vector, denoted as p_a and w_a, so $p_a = (2, 0, 2)$ and $w_a = (0, 2, 1)$.

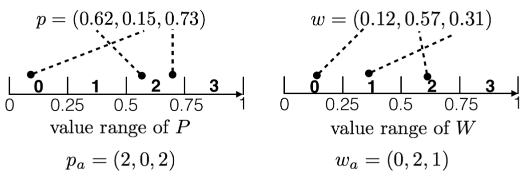

Figure 5: 4 partitions, allocating real values into approximate intervals and getting the approximate vector p_a and w_a.

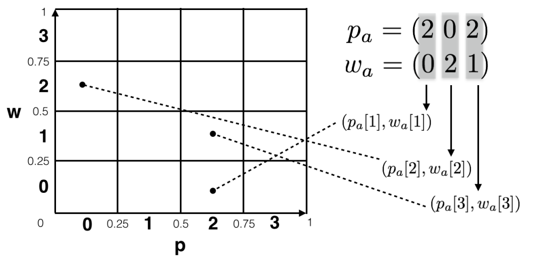

Figure 6: Mapping pairwise $(p[i], w[i])$ onto the 4×4 Grids.

Then we combined the range p and range w to form the grids. The purpose is that we want to map an arbitrary pair of $(p[i], w[i])$ into a certain grid, since the inner product $f(w, p)$ is the sum of pairwise multiplications of $(p[i], w[i])$. Figure 6 shows the 4×4 grids and the mapping $(p[i], w[i])$ onto grids. It is important to know that the same value range makes these grids can be re-used for mapping all pairs $(p[i], w[i])$, i.e., No matter how much *dimensions* the data has, only one Grid is sufficient.

Suppose that both the value range of p and w are divided into n partitions, and the step values of partition are represented by a (n+1)-element vector α_p for points and α_w for weights. In the example of Figure 5, $\alpha_p = \alpha_w = (0, 0.25, 0.5, 0.75, 1)$. The Grid-index is a 2-dimensional array and stores the multiplication results of all step values:

$$Grid[i][j] = \alpha_p[i] \cdot \alpha_w[j],\ i, j \in [0, n] \qquad (7)$$

5. 2 Upper and Lower Bounds of Score $f(w, p)$

We pre-calculate the approximate vectors set P_A and W_A for data P and W, respectively. For a $p_a \in P_A$ and a $w_a \in W_A$, we use the Grid-index to obtain the upper bound $U(w, p)]$, and lower bound $L(w, p)$:

$$L[f_w(p)] = \sum_{i=1}^{d} Grid[p_a[i]][w_a[i]] \qquad (8)$$

$$U[f_w(p)] = \sum_{i=1}^{d} Grid[p_a[i] + 1][w_a[i] + 1] \qquad (9)$$

Obviously, it costs only d times additions if the relation of scores can be confirmed by above upper or lower bounds.

Algorithm 1 GIA

```
Input: $P_A, W_A, P, W, Q, k$
Output:
 1: $buffer \leftarrow \emptyset$
 2: $kthRank \leftarrow \infty$
 3: for each $w_a \in W_A$ do
 4:     $Candidate \leftarrow \emptyset$
 5:     $Counter \leftarrow Domin.size$
 6:     Compute each $f(w, q)$ of $q \in Q$ and store in $Q_{scores}$
 7:     Sort $Q_{scores}$ in descending
 8:     for each $p_a \in (P_A\ /\ Domin\ )$ do
 9:         for $i$ to $|Q|$ do
10:             if $U[f(w, p)] \leqq Q_{scores}[i]$ then
11:                 $Counter \leftarrow Counter + |Q| - i$
12:                 if $counter == kthRank$ then
13:                     break to check next $w_a$
14:             if $L[f(w, p)] \leqq Q_{scores}[i] \leqq U[f(w, p)]$ then
15:                 $Candidate \leftarrow Candidate \cup \{p\}$
16:         if $p$ dominates $Q$ then
17:             $Domin \leftarrow Domin \cup \{p\}$
18:     Refine $Candidate$ and update $Counter$.
19: if $Counter \geqq kthRank$ then
20:     continue to next $w_a$
21: else
22:     $buffer.insert(w, Counter)$
23:     $kthRank \leftarrow$ the kth rank in $buffer$.
24: return buffer
```

5. 3 Grid-index Aggregate Reverse Rank Algorithm (GIA)

Algorithm 1 describes the proposed method GIA. It is a double looping framework that scans each $p_a \in P_A$ for each $w_a \in W_A$, and look up the Grid-index to avoiding computing. For each w_a, we use a *Counter* to record the aggregate rank $ARank(w, Q)$ (Line 4). In the inner loop(Line 8-17), the *Counter* will be update when the $f(w, q)$ is greater than a current point p. We carry out an optimization that calculates and sorts each q's score in advance (Line 6-7). We use Grid-index to obtain the $U[f(w, p)]$, if $U[f(w, p)]$ is smaller than a q's score, the *Counter* will increase by the remained number of q and we don't need to compare more (Line 10-11). If the rank relationship can not decide by the upper and lower from Grid-index, we add these kind of p into *Candidate* (Line 14-15), and we will refine the *Candidate* after scanning all p_a if it is necessary (Line 18). Another optimization is that "global dominating point", a p is a global dominating point if all $p[i] \leqq q[i]$ where $i = 1, 2, ..., d$, and we keep them into

Parameter	Values
Data dimensionality d	$3 \sim 20$, **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	$\sqrt[3]{\|P\|}$, $\sqrt[3]{\|W\|}$

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

a set *Domin* (Line 16-17). We will skip checking the global dominating points (Line 8) instead of initializing *Counter* by the size of *Domin* (Line 5). A k-element *buffer* is used to keep the top-k w's and their aggregate ranks for Q as the result of ARR query (Line 1, 22). The algorithm will break and start to check the next w_a when the *Counter* reaches *kthRank*, which is initialize as ∞ (Line 2) and update by the rank value of the last element in *buffer* (Line 23).

6. EXPERIMENT

We present the experimental evaluation of the SCAN, DTM and GIA algorithms for aggregate reverse k-rank. SCAN has the same strategy as that of GIA; the only difference is that SCAN computes the scores while GIA uses Grid-index. 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 [6,8].

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 $d = 6$, $|P|$=100K, $|W|$=100K, $k = 10$, $|Q| = 10$, 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.

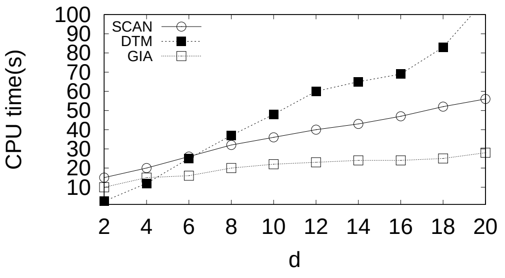

Figure 7: Experimental result on UN data, $|P| = 100\text{K}$, $|W| = 100\text{K}$, $|Q| = 10$, $k = 10$, $d = 2 \sim 20$.

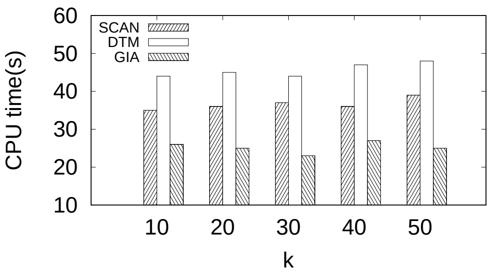

Figure 8: Experimental result on UN data, $|P| = 100\text{K}$, $|W| = 100\text{K}$, $|Q| = 10$, $d = 6$, $k = 10 \sim 50$.

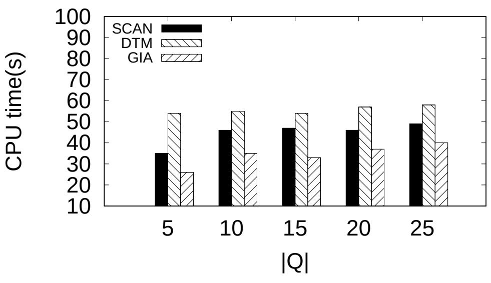

Figure 9: Experimental result on UN data, $|P| = 100\text{K}$, $|W| = 100\text{K}$, $k = 10$, $d = 6$, $|Q| = 5 \sim 25$.

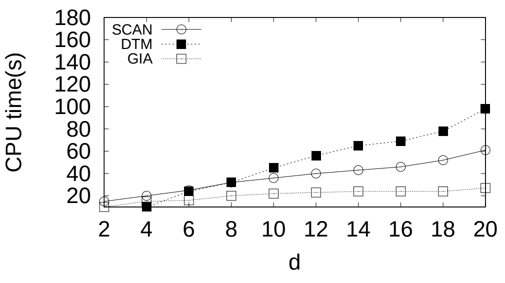

Figure 10: Experimental result on CL data, $|P| = 100\text{K}$, $|W| = 100\text{K}$, $|Q| = 10$, $k = 10$, $d = 2 \sim 20$.

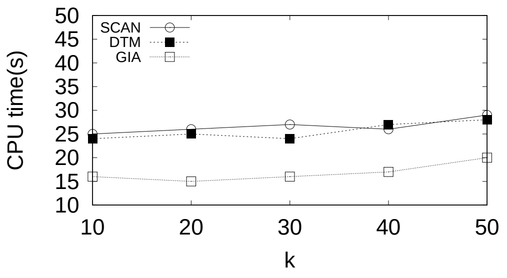

Figure 11: Experimental result on HOUSE data, $|P| = 100$K, $|W| = 100$K, $|Q| = 10$, $d = 6$, $k = 10 \sim 50$.

6.2 Experiment result

Un data. Figure 7 shows the experimental results on uniform distribution data sets on varying dimension d (from 3 to 20). According to the result, we can see that both SCAN and GIA becomes faster than DTM algorithm after six dimensions. GIA always faster than SCAN at least two times because it use Grid-index to filter. Figure 8 and Figure 9 show the CPU time of varying k (from 10 to 50) and varying $|Q|$ (from 5 to 25) respectively. All algorithm are insensitive to k and Q since $k \ll |P|$, $k \ll |W|$ and $|Q| \ll |P|$, $|Q| \ll |W|$.

CL data. Figure 10 shows the experimental results on Cluster data sets on varying dimension d (from 3-20). Notice that DTM take less time to finish querying than UN data because Cluster data is easy to index by R-tree. But GIA still becomes outperform DTM after eight dimensions.

HOUSE data. Figure 11 shows the performance of real data set HOUSE with different k (from 10 to 50). Clearly, GIA is faster than SCAN and DTM.

7. CONCLUSION

In marketing analysis, aggregate reverse rank query can be used to help manufacturers recognize their consumer base by matching their bundled products with user preferences. The state-of-the-art solution DTM is a tree-based algorithms, and are not designed to deal with high-dimensional data. In this paper, we proposed the Grid-index and the GIA algorithm to overcome the cost of high-dimensional computing. Experimental results confirmed the efficiency of the proposed algorithm when compared to the tree-based algorithms especially in high-dimensional cases.

In future work, we will focus on the optimization when the user preferences data $w \in W$ has many zero entry, i.e., when W is sparse. Since in practice, a user is normally interested in a few attributes of the products.

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] Dong, Y., Chen, H., Furuse, K., and Kitagawa, H. Aggregate reverse rank queries. In *Database and Expert Systems Applications - 27th International Conference, DEXA 2016, Porto, Portugal, September 5-8, 2016, Proceedings, Part II* (2016), pp. 87–101.

[4] 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.

[5] 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.

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

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

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

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

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

[11] 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.

[12] 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.

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