Reverse Top-k Queries
[DRAFT VERSION]

Akrivi Vlachou #, Christos Doulkeridis #, Yannis Kotidis *, Kjetil Nørvåg #

#*Department of Computer Science, NTNU, Trondheim, Norway*
{vlachou,cdoulk,noervaag}@idi.ntnu.no
**Department of Informatics, AUEB, Athens, Greece*
kotidis@aueb.gr

Abstract— **Rank-aware query processing has become essential for many applications that return to the user only the top-k objects based on the individual user's preferences. Top-k queries have been mainly studied from the perspective of the user, focusing primarily on efficient query processing. In this work, for the first time, we study top-k queries from the perspective of the product manufacturer. Given a potential product, which are the user preferences for which this product is in the top-k query result set? We identify a novel query type, namely *reverse top-k query*, that is essential for manufacturers to assess the potential market and impact of their products based on the competition. We formally define reverse top-k queries and introduce two versions of the query, namely monochromatic and bichromatic. We first provide a geometric interpretation of the monochromatic reverse top-k query in the solution space that helps to understand the reverse top-k query conceptually. Then, we study in more details the case of bichromatic reverse top-k query, which is more interesting for practical applications. Such a query, if computed in a straightforward manner, requires evaluating a top-k query for each user preference in the database, which is prohibitively expensive even for moderate datasets. In this paper, we present an efficient threshold-based algorithm that eliminates candidate user preferences, without processing the respective top-k queries. Furthermore, we introduce an indexing structure based on materialized reverse top-k views in order to speed up the computation of reverse top-k queries. Materialized reverse top-k views trade preprocessing cost for query speed up in a controllable manner. Our experimental evaluation demonstrates the efficiency of our techniques, which reduce the required number of top-k computations by 1 to 3 orders of magnitude.**

I. INTRODUCTION

Recently, the support of rank-aware query processing, has attracted much attention in the database research community. Top-k queries [1]–[10] retrieve only the k objects that best match the user preferences, thus avoiding huge and overwhelming result sets. Nowadays, most applications return to the user only a limited set of data points that are interesting for the user, therefore it is very important for a manufacturer that its products are returned in the highest ranked positions for as many different user preferences as possible. However, existing work studies only top-k queries from the perspective of customers that seek products matching their preferences. In this paper, we study top-k queries for business analysis, i.e. from the perspective of manufacturers who are interested in the impact of their products to customers, compared to their competitors existing products. The question that arises is "given a potential product, which are the user preferences for which this product is in the top-k query result set?". To this end, we propose *reverse top-k queries* and study two

different versions: monochromatic and bichromatic reverse top-k queries. In the former, there is no knowledge of user preferences and a manufacturer aims to estimate the impact a potential product would have on the market. In the latter, a dataset with user preferences is given and a reverse top-k query returns those preferences that rank a potential product highly. To the best of our knowledge, this is the first work that addresses this problem.
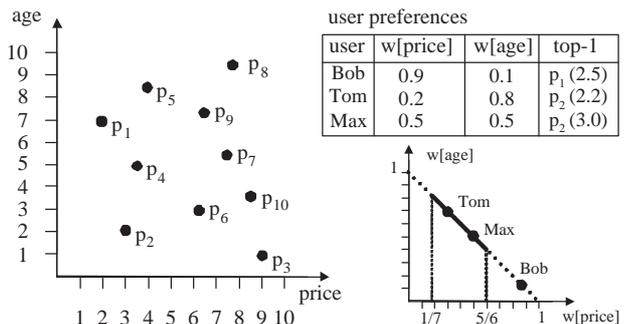

Fig. 1. Example of reverse top-k query.

A linear top-k query is defined by assigning a weight $w[i]$ to each of the attributes, expressing the importance of each attribute to the user. Without loss of generality, we assume that weights are normalized in [0,1] and $\sum w_i = 1$. This model is in agreement with the notion of preference [6], [8] and is widely adopted in related work. In the example of Figure 1, a database containing information about different cars is depicted. For each car, the price and the age are recorded and minimum values on each dimension are preferable. Different users have different preferences about a potential car and Figure 1 depicts also such a database of user preferences. For example, Bob prefers a cheap car, and does not care much about the age of the car. Therefore, the best choice (top-1) for Bob is the car p_1 which has the minimum score for the particular weights (namely 2.5). On the other hand, Tom prefers a newer car rather than a cheap car. Nevertheless, for both Tom and Max the best choice would be car p_2.

A reverse top-k query is defined by a given product p and returns the weighting vectors w for which p is in the top-k set. For example in Figure 1, the reverse top-1 result set of p_1 contains the weights $(0.9, 0.1)$ defined by Bob. Notice that for the car p_2, two weighting vectors belong to the reverse top-1 result set, namely the preferences of Tom and Max. In fact, all weighting vectors with $w[price]$ in the range of $[\frac{1}{7}, \frac{5}{6}]$

belong to the reverse top-1 result set of p_2. This segment of line $w[price] + w[age] = 1$ corresponds to the result set of the monochromatic reverse top-1 query (for $p=p_2$), whereas the set $\{(0.5, 0.5), (0.2, 0.8)\}$ is the result of the bichromatic reverse top-1 query for the given dataset of user preferences.

Conceptually, the solution space of reverse top-k queries is the space defined by the weights $w[price]$ and $w[age]$. Monochromatic reverse top-k queries return partitions of the solution space and are useful for business analysis and more particularly to estimate the impact of a product when no user preferences are given, but the distribution of them is known. In our example, under assumption of uniform distribution of user preferences, the impact in the market of the potential product p_2 can be estimated as $(\frac{5}{6} - \frac{1}{7}) \times 100\% = 69\%$. On the other hand, bichromatic reverse top-k queries have even wider applicability, as they identify users that are interested in a particular product, given a known set of user preferences. For instance, the best strategy for a profile-based marketing service would be to advertise car p_1 to Bob and car p_2 to Tom and Max. Notice that an empty result set for a product (i.e. car p_3) indicates that it is not interesting for customers based on their preferences. In practice, the bichromatic reverse top-k query can be used in practical applications and is easier to incorporate into a database management system, whereas the monochromatic mainly provides a geometric interpretation and helps to intuitively understand the problem.

Reverse top-k queries differ from reverse nearest neighbor (RNN) queries [11]. An RNN query retrieves the set of points having the query point as their nearest neighbor and there exists a monochromatic and a bichromatic version. In contrast to RNN queries, the reverse top-k query q finds the distance functions (in terms of weights) for which q would qualify as a k-nearest neighbor of the point positioned at the origin of the data space. Therefore, existing reverse nearest neighbor algorithms cannot be applied for reverse top-k queries. Reverse skyline queries [12] aim at identifying customers that are interested in a product, based on the dominance relationship. Nevertheless, user preferences are expressed as points with the same attributes as the products. In our case, user preferences are modeled in a more generic way (only in terms of weights) and they do not need to be uniquely mapped to a point in the data space.

To summarize, the main contributions of this paper are:

- We introduce a novel query type called reverse top-k query and present two versions, namely monochromatic and bichromatic. To the best of our knowledge, this is the first time that such queries are proposed.
- We analyze the geometrical properties for the two dimensional case of the monochromatic reverse top-k query and provide an algorithmic solution.
- We present an efficient and progressive threshold-based algorithm for computing bichromatic reverse top-k queries, which eagerly discards candidate user preferences, without the need to evaluate the associated top-k queries. Our algorithm consistently outperforms the brute force algorithm by 1 to 3 orders of magnitude.
- We present an indexing structure based on space parti-

tioning, which materializes reverse top-k views, in order to further improve reverse top-k query processing. The use of our index bounds the average cost of processing a bichromatic reverse top-k query in a straightforward manner.
- We conduct a thorough experimental evaluation that demonstrates the efficiency of our algorithms.

The rest of this paper is organized as follows: in Section II we formally define reverse top-k queries after providing the necessary preliminaries. In Section III, we study the geometrical properties of the two dimensional result set and propose an algorithm for monochromatic reverse top-k queries. Thereafter, in Section IV we present an efficient threshold-based algorithm for processing bichromatic reverse top-k queries for arbitrary data dimensionality. We introduce an indexing approach, based on materialized reverse top-k views in Section V, and discuss construction, usage and maintenance. The experimental evaluation is presented in Section VI. Then, Section VII reviews the related work and finally, in Section VIII, we conclude and discuss future research directions.

II. PROBLEM STATEMENT

In this section, we present the basics regarding top-k queries and then we proceed to define our problem statement.

A. Preliminaries

Given a data space D defined by a set of d dimensions $\{d_1, ..., d_d\}$ and a dataset S on D with cardinality $|S|$, a point $p \in S$ can be represented as $p = \{p[1], ..., p[d]\}$ where $p[i]$ is a value on dimension d_i. We assume that each dimension represents a numerical scoring attribute and therefore, the values $p[i]$ in any dimension d_i are numerical non-negative values that evaluate certain features of database objects. Furthermore, without loss of generality, we assume that smaller score values are preferable.

Top-k queries are defined based on a scoring function f that aggregates the individual scores into an overall scoring value, that in turn enables the ranking (ordering) of the data points. The most important and commonly used case of scoring functions is the weighted sum function, also called linear. Each dimension d_i has an associated query-dependent weight $w[i]$ indicating d_i's relative importance for the query. The aggregated score $f_w(p)$ for data point p is defined as a weighted sum of the individual scores: $f_w(p) = \sum_{i=1}^{d} w[i] \times p[i]$, where $w[i] \geq 0$ $(1 \leq i \leq d)$ and $\exists j$ such that $w[j] > 0$. The linear weighting function is increasingly monotone and it conveys the meaning that whenever the score of all dimensions of the point p is at least as good as another point p', then we expect that the overall score of p is at least as good as p'. Notice that assigning a zero weight to some dimensions leads to a top-k query referring only to a subset of the available features.

The result of a top-k query is a ranked list of the k objects with the best scoring values f_w. The weights indicate the user preferences and influence the ordering of the data objects and therefore the top-k result set. Consider for example the dataset depicted in Figure 2. By assigning a high weight to dimension x, point p_1 is the top-1 object, while if a low weight is used,
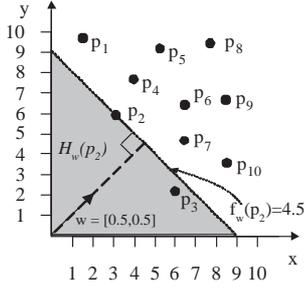
Fig. 2.   Top-k query.

(a) Data Space   (b) Solution Space.

Fig. 3.   Monochromatic reverse top-k query.

point p_3 becomes the top-1 object. A linear top-k query takes two parameters and can be expressed as $TOP_k(w)$, where w is a d-dimensional vector $w = \{w[1], ...w[d]\}$ that represents preference values.

Definition *(Top-k query)*: Given a positive integer k and a user-defined weighting vector w, the result set $TOP_k(w)$ of the top-k query, is a set of points such that $TOP_k(w) \subseteq S$, $|TOP_k(w)| = k$ and $\forall p_1, p_2 : p_1 \in TOP_k(w)$, $p_2 \in S - TOP_k(w)$ it holds that $f_w(p_1) \leq f_w(p_2)$.

In the Euclidean space a linear top-k query can be represented by a vector w. As discussed in [13] the magnitude of the query vector does not influence the query result, as long as the direction remains the same, i.e. representing the relative importance between different dimensions. Therefore, we make the assumption that $\sum_{i=1}^{d} w[i] = 1$.

There is a one-to-one correspondence between any weighting vector w and a hyperplane ℓ that crosses a point p. In a d-dimensional space, we call the $(d\text{-}1)$-dimensional hyperplane, which is perpendicular to vector w and contains a point p as the *query plane of w crossing p*, and denote it as $\ell_w(p)$. All points lying on the query plane $\ell_w(p)$, have the same scoring value equal to the score $f_w(p)$ of point p. Figure 2 depicts an example, where the query plane (equivalent to a query line in 2d) is perpendicular to the weighting vector $w = [0.5, 0.5]$. All points p_i lying on the query line have a score value $f_w(p_i) = f_w(p_2) = 4.5$. Furthermore, point p_2 is the top-2 object for the query $0.5 \times x + 0.5 \times y$. The ranking of a point p based on a weighting vector w is equal to the number of the points enclosed in the half-space defined by the query line (or $(d\text{-}1)$-dimensional query plane) that contains the origin of the data space. In the rest of the paper, we refer to this half-space as *query space of w defined by p* and denote it as $\mathcal{H}_w(p)$.

B. Definition of Reverse Top-k Queries

In this section, we formally define the monochromatic and the bichromatic reverse top-k query.

Definition *(Monochromatic Reverse top-k)*: Given a point q and a positive number k, as well as a dataset S, the result set of the monochromatic reverse top-k ($mRTOP_k(q)$) query of point q is the locus[1], i.e. a collection of d-dimensional points w_i, for which $\exists p \in TOP_k(w_i)$ such that $f_{w_i}(q) \leq f_{w_i}(p)$.

Definition *(Bichromatic Reverse top-k)*: Given a point q and a positive number k, as well as two datasets S and W, where S

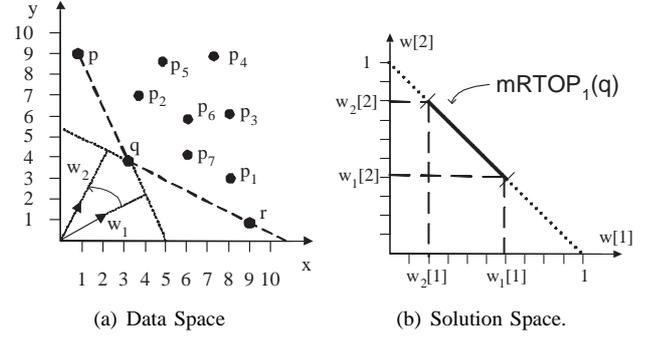[1]In mathematics, locus is the set of points satisfying a particular condition, often forming a curve of some sort.

represents data points and W is a data set containing different weighting vectors, a weighting vector $w_i \in W$ belongs to the bichromatic reverse top-k ($bRTOP_k(q)$) result set of q, if and only if $\exists p \in TOP_k(w_i)$ such that $f_{w_i}(q) \leq f_{w_i}(p)$.

For the sake of brevity, in the rest of this paper we denote a query point $q \in TOP_k(w_i)$, instead of $\exists p \in TOP_k(w_i)$ such that $f_{w_i}(q) \leq f_{w_i}(p)$. Consider for example the dataset depicted in Figure 2. For a query point $q=p_2$, the weighting vector w belongs to the reverse top-k, if the query space $\mathcal{H}_w(p_2)$ of w defined by p_2 (depicted as shadowed triangle) contains less than k points. The challenge is to find all the weighting vectors w_i that define query spaces $\mathcal{H}_{w_i}(q)$ containing less than k points. For the bichromatic version of the reverse top-k query, the result set contains a finite number of weighting vectors, while the monochromatic version aims to describe the parts of the solution space that satisfy the query.

III. MONOCHROMATIC REVERSE TOP-k QUERIES

Given a dataset S, a monochromatic reverse top-k query returns all weighting vectors w, for which query point $q \in TOP_k(w)$. Let us assume that W denotes the set of all valid assignments of w. Figure 3 shows the data and solution space of a 2d monochromatic reverse top-k query. Since $\sum_{i=1}^{d} w[i] = 1$ and $w[i] \in [0, 1]$, all valid weighting vectors of the reverse top-k query form the line $w[1]+w[2] = 1$ in the 2d solution space that is defined by the axis $w[1]$ and $w[2]$. Notice that it is not possible to enumerate all possible assignments of $w \in W$, since the number of possible vectors w is infinite. On the other hand, the solution space W can be split into a finite set of partitions W_i ($\bigcup W_i = W$, $\bigcap W_i = \emptyset$), such that query point q has the same ranking position for all the weighting vectors $w \in W_i$. Then, the result set of the monochromatic reverse top-k is a set of partitions W_i of the solution space W:

$$mRTOP_k(q) = \{W_i : \exists w_j \in W_i \wedge q \in TOP_k(w_j)\}$$

The main topic of this section is finding the partitions that form the result set of a monochromatic reverse top-k query. In the following, we focus on the 2-dimensional case. First, an example is given in order to provide some intuition about the problem. Then, we provide an algorithm for the monochromatic reverse top-k query.

A. Interpretation of Solution Space

Consider for example the dataset depicted in Figure 3(a). Only three points p, q and r belong to the top-1 result set
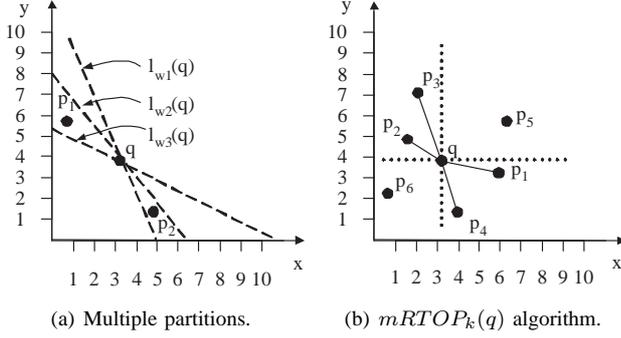
Fig. 4. Examples of $mRTOP_k(q)$ queries.

for any weighting vector, since these are the only points that belong to the convex hull [2]. There exists at least one weighting vector w_i for which $q \in TOP_1(w_i)$, and therefore at least one partition $W_i \in mRTOP_1(q)$. In order to determine the boundaries of the partition W_i, the line segments pq and qr have to be examined.

Let w_1 be the weighting vector that is perpendicular to pq, then it holds that $f_{w_1}(p)=f_{w_1}(q)$ and therefore p and q have exactly the same rank for the $TOP_1(w_1)$. Recall that the line $\ell_w(p)$ that is perpendicular to the weighting vector w and crosses p, defines the value of the scoring function and also the rank of point p according to w. For weighting vectors with smaller and larger angles than w_1, the relative order of p and q changes. If p had a lower rank than q for vectors with smaller angle than w_1, then for vectors with larger angle than w_1, point p has a higher rank. Since the relative order between p and q changes only once, there exists exactly one partition W_i, such that for all the weighting vectors $w \in W_i$ it holds that $q \in TOP_1(w)$.

The boundaries of the partition W_i are defined by the weighting vectors w_1, w_2 that are perpendicular to the line segments pq and qr respectively. All weighting vectors w for which the following inequality holds are in the reverse top-1 result set of q:

$$\frac{\lambda_{qr}}{\lambda_{qr}-1} \leq w[1] \leq \frac{\lambda_{pq}}{\lambda_{pq}-1}$$

where $\lambda_{pq} = \frac{q[2]-p[2]}{q[1]-p[1]}$ and $\lambda_{qr} = \frac{r[2]-q[2]}{r[1]-q[1]}$ are the slopes of lines pq and qr respectively. The above inequalities are derived by using the properties that $w_1 \perp pq$ and $w_2 \perp qr$. The result set of the monochromatic reverse top-1 query $mRTOP_1(q)$ is a segment (partition) of the line $w[1] + w[2] = 1$ in the 2-dimensional solution space defined by $w[1]$ and $w[2]$, as shown in Figure 3(b).

B. Monochromatic Reverse Top-k Algorithm

In our previous example, the result set $mRTOP_1(q)$ contains at most one partition W_i of W. However, for a reverse top-k query with $k > 1$, the result set may contain more than one non-adjacent partitions W_i. Consider for example the three data points in Figure 4(a) and assume we are interested to compute the $mRTOP_k(q)$ for $k=2$. Query point q is in the top-2 result set for both weighting vectors w_1 and w_3. However, when weighting vector w_2 is considered, with angle between w_1 and w_3, it is obvious that q no longer belongs to the top-2. Thus, in this small example, the monochromatic top-k query would return two non-adjacent partitions W_i.

Algorithm 1 Monochromatic Reverse top-k Algorithm.
```
 1: Input: S, q
 2: Output: mRTOP_k(q)
 3: W' ← {∅}, R ← {∅}, RES ← {∅}
 4: for (∀p_i ∈ S) do
 5:    if (q ⊀ p_i and p_i ⊀ q) then
 6:       w_i[1] ← λ_{p_i q}/(λ_{p_i q}−1), , w_i[2] ← 1 − w_i[1]
 7:       W' ← W' ∪ {w_i}
 8:    end if
 9: end for
10: sort W' based on w_i[1]
11: w_0 ← [0, 1], w_{|W'|+1} ← [1, 0]
12: R ← {p : p lies in H_{w_0}(q)}
13: k_w ← |R| //number of points in R
14: for (∀w_i ∈ W') do
15:    if (k_w ≤ k) then
16:       RES ← RES ∪ {(w_i, w_{i+1})}
17:    end if
18:    if (p_{i+1} ∈ R) then
19:       k_w ← k_w − 1
20:    else
21:       k_w ← k_w + 1
22:    end if
23: end for
24: return RES
```

Algorithm 1 describes the monochromatic reverse top-k algorithm. Data points that are dominated by q are always ranked after q for any weighting vector w, while points that dominate q are ranked before q for any weighting vector w. For example in Figure 4(b), p_5 is worse (ranked lower) than q, whereas p_6 is better (ranked higher) than q for any w. Points of the dataset that are neither dominated by nor dominate q are ranked higher than q for some weighting vectors and lower than q for other weighting vectors. Thus, our algorithm examines only such incomparable points $\{p_i\}$ to q (line 5), because they alter the rank of q. Fortunately, the weighting vector w_i for which the rank between q and a data point p_i changes, can be easily determined as the vector that is perpendicular to the line qp_i ($l_{w_i}(q)$). Consequently, we have to examine all lines[2] that pass through q and any other point p_i, which is incomparable to q. These lines define the boundaries of the partitions W_i, therefore the corresponding weighting vectors are kept in a list W' (line 7). Then, we identify the partitions for which q belongs to the top-k, by processing W'.

In Figure 4(b), after the sorting (line 10) the set W' is $\{w_1, w_2, w_3, w_4\}$ corresponding to the lines qp_1, qp_2, qp_3, qp_4 respectively. Then, vectors w_0 and w_5 are added to W'. For the first weighting vector w_0 all data points that lie in $H_{w_0}(q)$ are retrieved (line 12). Recall that the rank k_w of q with respect to w_0 is determined by the number of points contained in $H_{w_i}(q)$ (line 13). In our example, the set R is $\{p_4, p_6, p_1\}$ and therefore the rank of q is 4. The rank of q cannot change before w_1. If we assume that $k=3$, than for the first partition $W_0=[w_0, w_1]$ the rank of q is higher than k and the partition W_0 can be safely discarded. Therefore, the next partition is $W_1 = [w_1, w_2]$. Since $p_1 \in R$ (line 18), this

---

[2]This is similar to the approach in [8], which is used to compute a robust layered index.

means that in W_1 the relative ordering of the points p_1 and q changes and now the rank of q is 3. Therefore, W_1 is added to $mRTOP_3(q)$ (line 16). Similarly, we can compute the rank of q for all W_i. In our example, W_1 is the only partition that qualifies for the $mRTOP_3(q)$ result set. Thus, Algorithm 1 returns the monochromatic reverse top-k result set for any two dimensional dataset.

IV. BICHROMATIC REVERSE TOP-K QUERIES

For a bichromatic reverse top-k query, two datasets S and W are given, where S contains the data points and W the different weighting vectors that represent user preferences. Then, the aim is to find all weighting vectors $w_i \in W$ such that the query point $q \in TOP_k(w_i)$.

A brute force (naive) approach is to process a top-k query for each $w_i \in W$ and examine whether q belongs to $TOP_k(w_i)$. Obviously, this approach induces high processing cost, as it requires one top-k query evaluation for each weighting vector w_i. As the number of potential weighting vectors w_i in the dataset W may be high (comparable to the size of the dataset $|S|$), this approach is prohibitively expensive and does not scale. In the sequel, we present a threshold-based algorithm (called RTA, **R**everse top-k **T**hreshold **A**lgorithm) for bichromatic reverse top-k, which discards weighting vectors that cannot contribute to the result set $bRTOP_k(q)$, without evaluating the corresponding top-k queries.

A. Threshold-based Algorithm (RTA)

Our algorithm exploits already computed top-k results to avoid evaluating weighting vectors that cannot be in the reverse top-k result set. The goal is to reduce the number of top-k query evaluations, based on the observation that top-k queries defined by similar weighting vectors return similar result sets [6]. Therefore, in each repetition a threshold is set based on to previously computed top-k result sets, in order to discard the next weighting vectors without top-k query evaluation.

As the aim is to examine similar weighting vectors in consecutive steps, the weighting vectors W are ordered based on their pairwise similarity. We measure the similarity between two vectors using the cosine similarity and the goal is to maximize the cosine similarity of all consecutive weighting vector pairs. To achieve an acceptable solution without overwhelming computational overhead, the weighting vectors are ordered based a simple strategy. The first weighting vector w_1 is the most similar vector to the diagonal vector of the space. Thereafter, the most similar weighting vector w_{i+1} to the previous vector w_i is examined. Notice that the ordering of the weighting vectors takes place during the initialization phase of the algorithm, and is not affected by the query point.

Algorithm 2 formally describes the RTA algorithm for processing a bichromatic reverse top-k query. Initially, RTA computes the top-k result $TOP_k(w_i)$ for the first weighting vector (line 7). Notice that in the first iteration we cannot avoid evaluating a top-k query, as the threshold cannot be set yet. The k data points that belong to the result set $TOP_k(w_i)$ are kept in a main memory buffer. Given a set of points P, we denote as $max\{f_{w_i}(P)\}$ the maximum value of all score values $f_{w_i}(p_j)$, $p_j \in P$, which means that

---

Algorithm 2 RTA: Reverse Top-k Threshold Algorithm.
1: **Input:** S, W, q, k
2: **Output:** $bRTOP_k(q)$
3: $W' \leftarrow \{\emptyset\}$, $buffer \leftarrow \{\emptyset\}$
4: $threshold \leftarrow \infty$
5: **for** (each $w_i \in W$) **do**
6:    **if** ($f_{w_i}(q) \leq threshold$) **then**
7:       $buffer \leftarrow TOP_k(w_i)$
8:       **if** ($f_{w_i}(q) \leq max\{f_{w_i}(buffer)\}$) **then**
9:          $W' \leftarrow W' \cup \{w_i\}$
10:      **end if**
11:   **end if**
12:    $threshold \leftarrow max\{f_{w_{i+1}}(buffer)\}$
13: **end for**
14: **return** W'

$max\{f_{w_i}(P)\} \geq f_{w_i}(p_j)$, $\forall p_j \in P$. The score $f_{w_i}(q)$ of query point q based on vector w_i is computed and compared against the maximum f_{w_i} value of all points in the buffer, denoted as $max\{f_{w_i}(buffer)\}$ (line 8). This maximum score defines the threshold value. If the score $f_{w_i}(q)$ is not greater than $max\{f_{w_i}(buffer)\}$, then w_i is added to the result set (line 9). Before the next iteration of the algorithm, we take the next weighting vector (w_{i+1}) and we set as threshold value the maximum score of any point in the buffer based on this new vector w_i (line 12). Then the condition of line 7 is tested, so if the score $f_{w_i}(q)$ is larger than the threshold, then we can safely discard w_i. Otherwise, we have to evaluate the top-k query for the vector w_i, in order to determine whether w_i belongs to the reverse top-k result. Therefore, we pose again a top-k query on dataset S and we update the main memory buffer with the new result set $TOP_k(w_i)$. In each iteration, the k points of the previously processed top-k query are kept in the buffer. Notice that the size of the buffer (k) is limited, since queries with small k values are commonly used in practice. The algorithm terminates when all weighting vectors have been evaluated or discarded.

Correctness of the algorithm: Let $w \in bRTOP_k(q)$ be a weighting vector that is falsely discarded without a top-k evaluation. Then, based on the definition of the reverse top-k query, $\exists p \in TOP_k(w)$ such that $f_w(q) \leq f_w(p)$. Let p_i, $1 \leq i \leq k$ be the points in the buffer, then based on the threshold $\forall p_i$: $f_w(p_i) < f_w(q)$. Therefore, $f_w(p_i) < f_w(q) \leq f_w(p)$, $\forall p_i$. This means that $p \notin TOP_k(w)$, which leads to a contradiction. $\square$

In worst case, the algorithm has to process $|W|$ top-k queries, hence the algorithm degenerates to the brute force algorithm. However, in the average case the algorithm returns the correct result by evaluating much less than $|W|$ top-k queries, which is verified also in the experimental evaluation. On the other hand, RTA has to evaluate at least $|bRTOP_k(q)|$ top-k queries, since no weighting vector w_i can be added in the result set without evaluating the respective top-k query.

B. RTA Example

In order to provide an intuitive example of RTA, consider the dataset consisting of points p_i, a dataset $W = \{w_1, w_2, w_3\}$, as well as two potential query points q and q', depicted in Figure 5. Let us assume that $k=2$ and the first
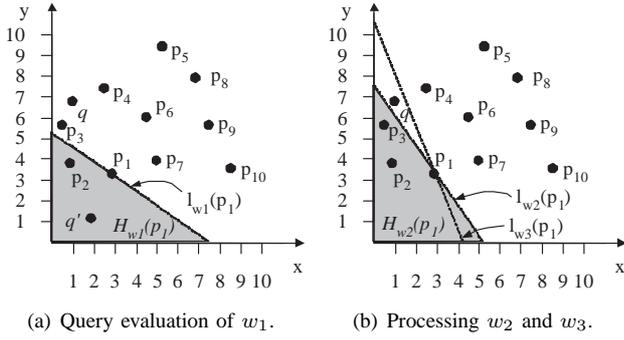
(a) Query evaluation of w_1.　　(b) Processing w_2 and w_3.

Fig. 5.　Example of bichromatic algorithm (RTA).

(a) Grid partitioning.　　(b) Cell example.

Fig. 6.　Example of grid-based algorithm.

examined weighting vector is w_1. As depicted in Figure 5(a), RTA computes the top-2 query ($TOP_2(w_1)$) and finds that the top-2 data point for w_1 is p_1. Points p_1 and p_2 are enclosed in the query space $\mathcal{H}_{w_1}(p_1)$ (depicted as gray triangle) and those points are kept in the buffer $\{p_2, p_1\}$. If the query point is q', then it would be enclosed in $\mathcal{H}_{w_1}(p_1)$ and w_1 would belong to $bRTOP_k(q')$. Now consider that the query is q, so it is not enclosed in $\mathcal{H}_{w_1}(p_1)$, therefore w_1 does not belong to the $bRTOP_k(q)$. In the next step (Figure 5(b)), the most similar weighting vector to w_1, namely w_2, is examined and the threshold is set based on the query line of w_2 crossing p_1, depicted as the gray triangle ($\mathcal{H}_{w_2}(p_1)$). Since q is not enclosed in $\mathcal{H}_{w_2}(p_1)$, the weighting vector w_2 is safely discarded, without further processing. Notice that $\mathcal{H}_{w_2}(p_1)$ contains at least 2 data points (in this example 3), and this explains why w_2 can be safely discarded. When the next vector w_3 is considered, q is enclosed in $\mathcal{H}_{w_3}(p_1)$, therefore the result set $TOP_2(w_3)$ has to be retrieved, and the buffer now contains $\{p_2, p_3\}$. Notice that the score value of q is not better than the score value of p_3 that is the top-2 data point of this query, so the weighting vector w_3 is not added to the reverse top-2 result set of q. Thus, none of the 3 weighting vectors belongs to the result of $bRTOP_2(q)$.

V. MATERIALIZED REVERSE TOP-k VIEWS

In this section, we present an indexing structure ($RTOP\text{-}Grid$) based on space partitioning, which materializes reverse top-k views for efficient processing of bichromatic reverse top-k queries. First, we present an example that explains how RTOP-Grid improves the performance of reverse top-k queries, by further reducing the required number of top-k query evaluations. Then, we describe in detail the reverse top-k algorithm based on RTOP-Grid. Afterwards, we clarify the details on how the construction of the RTOP-Grid can be accomplished in an efficient manner. Finally, we generalize our approach for arbitrary k values and discuss updates.

A. Motivating Example

Let us assume that we have a grid-based space partitioning of the data space. The grid consists of disjoined data space partitions, also called *cells*. Each cell C_i is defined by its lower left corner C_i^L and upper right corner C_i^U. For each cell C_i and a given value k, a reverse top-k query for each corner C_i^L and C_i^U is evaluated and the result set is stored. More particularly, the resulting weighting vectors w_z are maintained in a list associated with the corresponding corner,
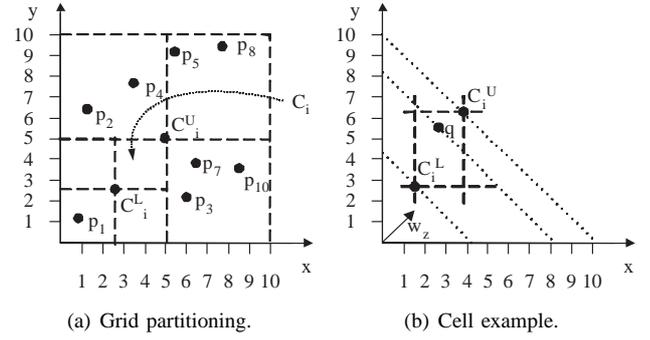
for example for the lower left corner C_i^L we define as $L_i^L = \{w_z \in bRTOP_k(C_i^L)\}$. Analogously, L_i^U is defined. During the reverse top-k evaluation, each grid corner is considered as a query point and the query is evaluated (using Algorithm 2) against the dataset S, ignoring the remaining grid corners. The list L_i^L (L_i^U) associated with a corner C_i^L (C_i^U) of C_i is considered as a materialized reverse top-k view, as it contains all weighting vectors that would be the result of a query $bRTOP_k(C_i^L)$ ($bRTOP_k(C_i^U)$). Henceforth, we refer to the lists of weighting vectors of a cell as materialized views.

For example, consider the grid depicted in Figure 6(a). During query processing we can exploit the information associated with the corners of the cells, in order to restrict the number of weighting vectors that need to be examined by our reverse top-k algorithm. Given a query point q, the cell C_i which encloses query point q is determined. It is obvious that for any w_z that does not exist in materialized view L_i^L, w_z cannot be in the reverse top-k of q. This is because C_i^L dominates q, thus for any w_z, query point q has a higher score than the corner C_i^L. Therefore, if the corner C_i^L does not qualify for the top-k result set, neither can point q. For example, consider Figure 6(b). If $w_z \notin L_i^L$, then the query space $\mathcal{H}_{w_z}(C_i^L)$ defined by the query plane $\ell_{w_z}(C_i^L)$, contains more than k data points. Consequently, also the query space $\mathcal{H}_{w_z}(q)$ contains more than k points, and therefore $q \notin TOP_k(w_z)$.

On the other hand, if a weighting vector w_z belongs to materialized view L_i^U, then w_z is definitely in the reverse top-k result of q. To explain this better, consider Figure 6(b) and that w_z belongs to the reverse top-k result set of C_i^U, i.e. $w_z \in L_i^U$. This means that less than k data points exist in the query space $\mathcal{H}_{w_z}(C_i^U)$. Therefore, since q is enclosed in C_i, then it is also in the top-k result, independently of q's exact position in the cell. Notice that a weighting vector w_z that belongs to L_i^U, also belongs to L_i^L.

Only for weighting vectors that are in L_i^L but not in L_i^U we have to examine the exact ranking of q based on its position. Essentially, this restricts the input of Algorithm 2, which is used to compute the $bRTOP_k(q)$, to consider weighting vectors only from the set $L_i^L - L_i^U$, rather than W.

B. Grid-based Reverse Top-k Algorithm (GRTA)

Algorithm 3 formally describes how a bichromatic reverse top-k query is processed using the grid-based materialized views. Initially, the cell C_i that encloses q is determined (line 4). Then, each weighting vector $w_z \in L_i^L$ is further examined (line 5). If w_z belongs also to L_i^U (line 6), then we are certain

Algorithm 3 GRTA: Grid-based Reverse top-k Algorithm.

1: **Input:** S, q, k
2: **Output:** $bRTOP_k(q)$
3: $W' \leftarrow \{\emptyset\}$, $W'' \leftarrow \{\emptyset\}$, $W_{cand} \leftarrow \{\emptyset\}$
4: Find cell C_i that encloses q
5: **for** $(\forall w_z \in L_i^L)$ **do**
6:    **if** $(w_z \in L_i^U)$ **then**
7:       $W' \leftarrow W' \cup \{w_z\}$
8:    **else**
9:       $W_{cand} \leftarrow W_{cand} \cup \{w_z\}$
10:    **end if**
11: **end for**
12: $W'' \leftarrow \text{RTA}(S, W_{cand}, q, k)$
13: **return** $\{W' \cup W''\}$

Algorithm 4 Construction of RTOP-Grid.

1: **Input:** S, W, k, $Limit$
2: **Output:** RTOP-Grid
3: Create cell C_0 that covers the universe
4: $L_0^L \leftarrow \text{RTA}(S, W, C_0^L, k)$
5: $L_0^U \leftarrow \text{RTA}(S, W, C_0^U, k)$
6: $RES \leftarrow \{C_0\}$
7: $cntCells \leftarrow 1$
8: **while** $(cntCells < Limit)$ **do**
9:    Find cell C_i with maximum $COST_{C_i}$
10:   Split C_i into C_1 and C_2 based on d_j
11:   $L_1^L \leftarrow L_i^L$
12:   $L_1^U \leftarrow \text{GRTA}(S, C_1^U, k)$
13:   $L_2^L \leftarrow \text{GRTA}(S, C_2^L, k)$
14:   $L_2^U \leftarrow L_i^U$
15:   $RES \leftarrow RES - \{C_i\}$
16:   $RES \leftarrow RES \cup \{C_1, C_2\}$
17:   $cntCells \leftarrow cntCells + 1$
18: **end while**
19: **return** RES

that w_z belongs to the reverse top-k result of query point q, so we add w_z to list W' (line 7) that contains the results. If w_z does not belong to L_i^U, then w_z is added (line 9) to the set of candidate weighting vectors W_{cand} that need to be evaluated. Finally, we invoke Algorithm 2 on the set of candidate weighting vectors W_{cand} (line 12) and some of them are returned as results denoted as W''. The weighting vectors that belong to the union of W' and W'' constitute the results of the GRTA algorithm (line 13).

As already discussed, the cost of RTA (Algorithm 2) depends mainly on the number of top-k evaluations. This number is related to the cardinality of the dataset W, which is given as an input to the algorithm. Therefore, by using the grid-based materialization, the number of weighting vectors that need to be examined in order to retrieve the reverse top-k result is restricted, since Algorithm 2 takes as input a limited set of weighting vectors W_{cand}, instead of the entire set W. In particular, the upper bound of top-k evaluations for different weighting vectors is $|L_i^L| - |L_i^U|$, which is the number of evaluations required by the brute force algorithm. Of course, RTA reduces even more this number, by discarding weighting vectors based on already computed results. However, the exact savings in terms of discarded weighting vectors also depend on the construction algorithm and the quality of the resulting grid, as will be shown presently.

C. RTOP-Grid Construction

In this section, we discuss the construction algorithm of RTOP-Grid. In our approach, the grid-based space partitioning occurs recursively, starting by a single cell that covers the entire universe. We take into consideration three different subproblems. First, we develop a cost-based heuristic for deciding which cell C_i to split. Secondly, we accomplish efficient computation of the views L_i^L and L_i^U, by using a results sharing approach. Finally, we propose different strategies for establishing the stopping condition of the cell division process.

Given a cell C_i and a query point q enclosed in C_i, the performance of reverse top-k query depends mainly on the number of evaluated top-k queries, which in turn depends on the number of weighting vectors in the views L_i^L and L_i^U. Therefore, it is very important that the splitting strategy of the construction algorithm splits first the most costly cells, i.e. the cells that may lead to many top-k evaluations. We define the

notion of *cost* for a cell C_i as the probability that a query point is enclosed in a cell multiplied by the number of top-k query evaluations necessary for processing the query in C_i. Assume that $f(q[1], q[2], ..., q[d]) \equiv f(q)$ denotes the density function describing the distribution of the d variables corresponding to the dimensions of the query points. Then, the expected cost of a cell C_i can be estimated as:

$$COST_{C_i} = (|L_i^L| - |L_i^U|) \int_{C_i} f(q) \tag{1}$$

In the case of uniform query distribution, the integral of Equation 1 can be replaced by the fraction of the volume of the space covered by the cell (normalized volume $\frac{V(C_i)}{V_D}$).

Given a RTOP-Grid index, we define the average number of top-k query evaluations that are necessary for processing a reverse top-k query as a quality measure of RTOP-Grid, which can be expressed as the sum of the costs of all cells:

$$COST_{RTOP-Grid} = \sum_{\forall i} COST_{C_i} \tag{2}$$

The cost function insinuates that the cost of a particular cell adds up to the total cost of the grid, only if a query point is actually enclosed in the cell. Equation 2 is the average cost of processing a reverse top-k query, in terms of top-k evaluations for a given RTOP-Grid, because it contains the probability that a query is enclosed in a cell. Furthermore, the estimated cost is an upper bound of the actual cost, since RTA needs even fewer top-k evaluations than $|L_i^L| - |L_i^U|$. The splitting employed in the RTOP-Grid construction algorithm aims at minimizing the aforementioned cost function. Thus, the construction algorithm splits the cell with the maximum $COST_{C_i}$ value.

Algorithm 4 describes the construction of RTOP-Grid. Assuming initially a single cell C_0 covering the entire universe (line 3), the algorithm starts by computing the materialized views of the lower and upper corner of the universe (lines 4,5). In order to process the reverse top-k query for each cell's corners efficiently, the RTA algorithm is employed. In each iteration, the algorithm picks a cell C_i to be split, which is

the cell C_i with the maximum $COST_{C_i}$, according to our splitting strategy (line 9). Then, two new cells C_1 and C_2 are created (line 10) by selecting a dimension in a round robin fashion, which is used to divide the cell in two parts. Consequently, the materialized views of the new cells C_1 and C_2 are computed. Our algorithm employs result sharing in two ways. First, it is obvious that L_1^L and L_2^U equals to L_i^L and L_i^U respectively (lines 11,14), and these materialized views do not have to be recomputed. Whenever a reverse top-k query for each cell's corners needs to be computed, GRTA is employed (lines 12,13) on the currently constructed RTOP-Grid. Therefore, the algorithm takes into account the views of the existing cells to restrict the weighting vectors that need to be examined and the top-k queries that have to be evaluated. This is the second way result sharing is used, namely to efficiently compute the necessary materialized views. Finally, cell C_i is removed from the RTOP-Grid, whereas cells C_1 and C_2 that cover the removed cell are added (lines 15,16). The algorithm continues to iterate, until the stopping condition that ceases splitting of cells is satisfied (line 8).

As regards the stopping condition, two different strategies are used, each controlling the cost of a different parameter, namely storage requirements and query processing performance. Hence, two different strategies are employed:

- *Space-bounded:* In order to restrict the construction and storage cost, the algorithm stops when a specific number of grid cells (given as input) are created. Algorithm 4 describes this strategy (line 8).
- *Guaranteed cost:* This strategy focuses on query processing cost, rather than construction cost, and aims at setting a bound on the average number of required top-k evaluations. Cells are split as long as the quality of the RTOP-Grid, has not reached the bound (given as input). The quality is measured by means of Equation 2. Therefore, the stopping condition of Algorithm 4 (line 8) is modified as follows:
  $COST_{RTOP-Grid} \leq Limit.$

In our experimental evaluation, we also examine a straightforward approach, namely UNIFORM, where the algorithm decides to split the cell that has the largest volume, without using the cost function. The stopping condition follows the space-bounded strategy, i.e. splitting stops when a specified number of cells are created.

D. Supporting Arbitrary k Values

In this section, we generalize our approach to support reverse top-k queries for arbitrary values of k, using a common RTOP-Grid. Given an upper limit K_{max}, the RTOP-Grid is constructed for K_{max} and additional information is stored that enables processing queries for any k value ($k \leq K_{max}$). For each weighting vector w_z, the rank of the cell corner, i.e. the minimum k for which the corner is in the top-k result set of w_z, is additionally maintained. Thus, the materialized view can be described as $L_i^L = \{(w_z, k_z^L)\}$ and $L_i^U = \{(w_z, k_z^U)\}$.

Algorithm 3 can be adjusted to process reverse top-k queries over a grid constructed for arbitrary $k \leq K_{max}$. First, the cell C_i that encloses q is determined. Then, the

weighting vectors that are contained in L_i^L are examined, while weighting vectors that are not in L_i^L cannot contribute to the reverse top-k result set of q. For any $w_z \in L_i^L$, the following cases are distinguished (the following code replaces lines 6-10 of Algorithm 3):

```
IF (k_z^L ≤ k) THEN
    IF (w_z ∈ L_i^U and k_z^U ≤ k)
    THEN
        W' ← W' ∪ {w_z}
    ELSE
        W_cand ← W_cand ∪ {w_z}
```

E. Updates

Updates that occur either in W or S affect the materialized reverse top-k views, therefore they should be supported efficiently. In case of insertion of a new weighting vector w_{ins}, we need to progressively examine the corners of the grid, starting from the origin of the data space. If a corner C_i^L (C_i^U) does not qualify as top-k object for w_{ins}, then we can safely discard all corners dominated by C_i^L (C_i^U). Deletion of an existing weighting vector w_{del} is simple, as it requires removal of w_{del} from the lists of any corner of the grid. Notice that again the corners of the grid can be examined progressively, thus avoiding processing of dominated corners.

Insertion of a data point p_{ins} is more costly, since only grid corners that dominate p_{ins} are discarded. For the remaining corners, we cannot avoid computing the reverse top-k query. However, GRTA can be used and only weighting vectors that belong to the materialized views of the cell corner have to be evaluated, since no weighting vectors can be added, but only some of them may be removed from the materialized view. Similarly a data point p_{del} that is removed from the dataset probably influences all dominated cell corners, therefore we need to recompute the materialized views for them, since new weighting vectors may have to be added.

VI. EXPERIMENTAL EVALUATION

In this section, we present an extensive experimental evaluation of reverse top-k queries. All algorithms are implemented in Java and the simulations run on a 3GHz Dual Core AMD processor equipped with 2GB RAM. The block size is 8KB. We focus on the evaluation of the bichromatic reverse top-k query, as it is most useful for practical applications.

As far as the dataset S is concerned, both real (RE) and synthetic data collections, namely uniform (UN), correlated (CO) and anticorrelated (AC), are used. For the uniform dataset, all attribute values are generated independently using a uniform distribution. The anticorrelated dataset is generated by selecting a plane perpendicular to the diagonal of the data space using a normal distribution, and within the plane each attribute value follows a uniform distribution. Similarly, for the correlated dataset, first a plane perpendicular to the diagonal of the data space is selected by using a normal distribution and within the plane, each attribute value is generated using a normal distribution. We also use two real datasets. NBA consists of 17265 5-dimensional tuples, representing a player's
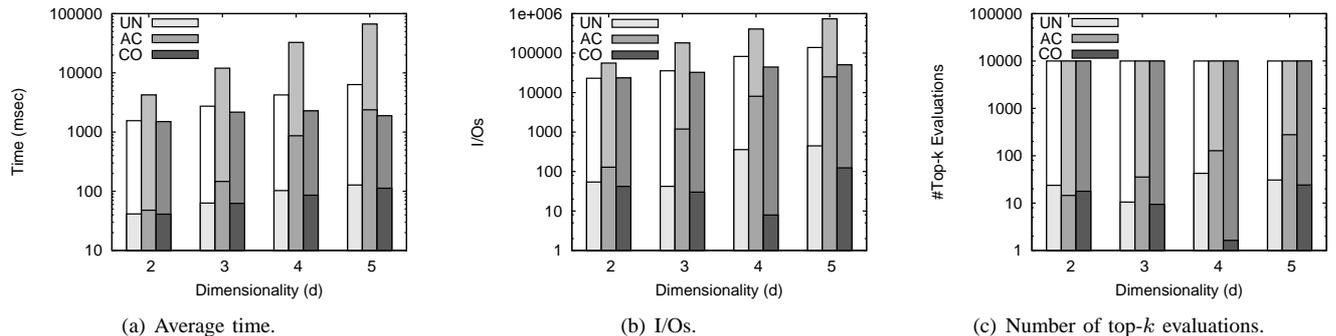
(a) Average time. (b) I/Os. (c) Number of top-k evaluations.

Fig. 7. Performance of RTA for varying d [naive (outer bar) vs. RTA (inner bar)].

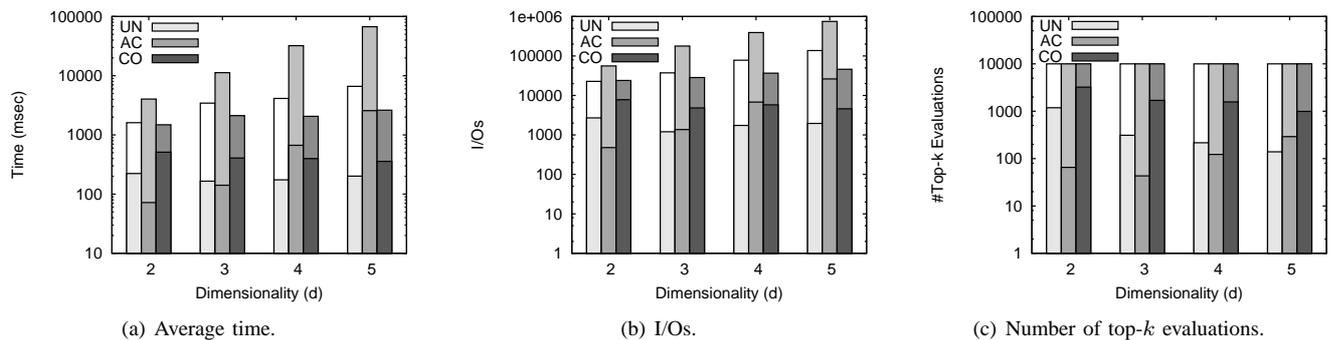(a) Average time. (b) I/Os. (c) Number of top-k evaluations.

Fig. 8. Performance of RTA for varying d for k-skyband queries [naive (outer bar) vs. RTA (inner bar)].

performance per year. The attributes are average values of: number of points scored, rebounds, assists, steals and blocks. HOUSE (Household) consists of 127930 6-dimensional tuples, representing the percentage of an American family's annual income spent on 6 types of expenditure: gas, electricity, water, heating, insurance, and property tax.

For the dataset W of the weighting vectors, two different data distributions are examined, namely uniform (UN) and clustered (CL). For the clustered dataset W, first C_W cluster centroids that belong to the $(d$-1$)$-dimensional hyperplane defined by $\sum w_i = 1$ are selected randomly. Then, each coordinate is generated on the $(d$-1$)$-dimensional hyperplane by following a normal distribution on each axis with variance σ_W^2, and a mean equal to the corresponding coordinate of the centroid. We conduct experiments varying the dimensionality (2-5), the cardinality (10k-100k) of the dataset S and cardinality (5k-15k) of the dataset W.

We evaluate the performance of RTA against an alternative technique that evaluates a top-k query for each weight in the dataset W. In particular, the dataset S is indexed by an RTree and top-k processing is performed using a state-of-the-art branch-and-bound algorithm. We refer to this algorithm as naive. Our metrics include: a) the time (wall-clock time) required by each algorithm, b) the I/Os used, and c) the number of top-k evaluations conducted. We also investigate the performance benefits that RTOP-Grid attains over RTA. We present average values over the 1000 queries in all cases. Notice that we do not measure the I/Os that occur by reading W, since this is the same for every method, assuming sequential scan on the dataset W.

A. Performance Evaluation of RTA

In Figure 7, we study the behavior of RTA for increasing dimensionality d, for various distributions (UN,AC,CO) of dataset S and uniform weights W. We use $|S|$=10k, $|W|$=10k, top-k=10 and 1000 random queries that follow the data distribution. Notice that the y-axis is in logarithmic scale. In the bar charts, each of the three bars (for a specific dimensionality) represents a dataset: UN, AC, and CO respectively. The total length of the bar represents the performance of naive, while the inner dark-colored bar depicts the performance of RTA. Regarding average time, RTA is 2 orders of magnitude better than naive, in all examined data distributions. In terms of I/Os, again RTA outperforms naive by 1 to 3 orders of magnitude, while larger savings are obtained for datasets UN and CO. The reason behind RTA's superiority is clearly demonstrated in Figure 7(c), where the average number of top-k evaluations necessary for computing a bichromatic reverse top-k query is shown. The threshold employed by RTA reduces significantly the number of top-k evaluations, saving around 1.5 to 3 orders of magnitude compared to naive. Notice that naive requires $|W|$ (=10k) top-k query evaluations to compute the result, regardless of data distribution.

An interesting observation is that only a small percentage (around 2%) of the queries actually return non-empty result sets. This is due to the fact that queries are generated following the data distribution, therefore many queries are not in the top-k result for any weighting vector. An important feature of our algorithm is the fact that RTA can efficiently process such queries. Thus, RTA processes reverse top-k queries that have a small or empty result set quickly, because the threshold employed eliminates candidate weighting vectors, often requiring
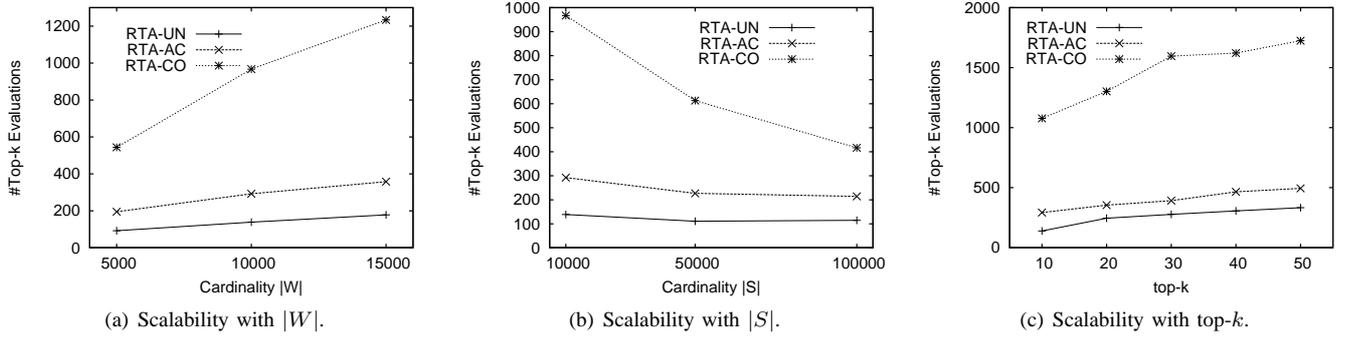
(a) Scalability with $|W|$.

(b) Scalability with $|S|$.

(c) Scalability with top-k.

Fig. 9.   Scalability study of RTA.

(a) Clustered dataset W.

(b) Real dataset (NBA).
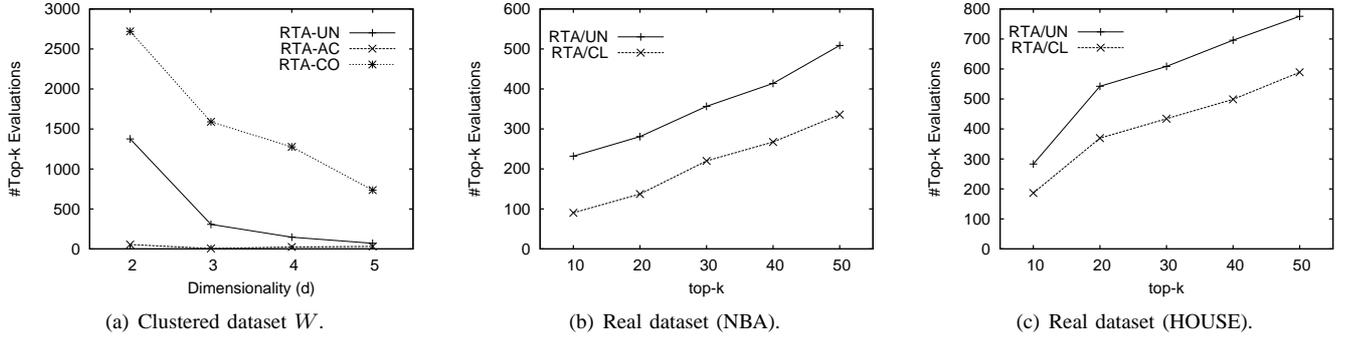
(c) Real dataset (HOUSE).

Fig. 10.   Performance of RTA for clustered weights W and for real data (NBA and HOUSE).

only one top-k evaluation. In contrast, naive does not have this ability and computes $|W|$ top-k queries for all queries. Notice that reverse top-k queries that produce empty result sets are also very informative for a product manufacturer, since they indicate that the particular product is probably not popular for any customer, compared to their competitors' products.

Nevertheless, we also employ a different method to generate queries and present the corresponding results. In order to increase the probability that a query point belongs to a top-k result, we pick random query points from the k-skyband of the dataset. Obviously, these query points are more likely to produce non-empty reverse top-k results. This query workload corresponds to queries about products that seem popular to costumers, and manufacturers are expected to pose such queries with high probability. Figure 8 depicts the results obtained by using k-skyband queries for the same experimental setup depicted in Figure 7. Although we witness a small deterioration in the results of RTA, our algorithm consistently outperforms naive by 1 to 2 orders of magnitude. Some interesting observations can be made by studying Figure 8(c). First, we notice that the correlated dataset requires more top-k evaluations caused by the fact that the cardinality of $bRTOP_k(q)$ is high. The reason is that the k-skyband of a correlated dataset contains points that are close to the origin of the data space, and therefore such points are in the top-k for many weighting vectors. Second, we observe a decreasing tendency as dimensionality increases, which seems counterintuitive at first. However, this is because again the cardinality of $bRTOP_k(q)$ decreases as the dimensionality increases. For the rest of our experiments, we use k-skyband queries and we do not show the results of naive, as it performs consistently worse than RTA by few orders of magnitude.

Thereafter, we perform a scalability study of RTA by varying several parameters in Figure 9. We use as metric the number of top-k evaluations, as it is the dominant factor for the performance of RTA. First, we increase the cardinality of W and study the performance of RTA for different data distributions of S (Figure 9(a)). We fix the remaining parameters to $|S|$=10k, d=5 and top-k=10. In general, datasets W of higher cardinality demand more top-k evaluations. However, we observe that RTA is highly efficient, especially for the costly CO dataset. For instance, for $|W|$=5k, RTA needs on average 544 top-k evaluations, while the average mandatory cost is 459 (this is the number of queries that cannot be avoided, also equal to the average size of the result set). This shows that out of 5000 query evaluations (100%), RTA needs only 544 (10.88%), which is only marginally more than the mandatory 459 (9.18%), thus RTA saves 89.12% of the cost.

In Figure 9(b), we set $|W|$=10k and gradually increase the cardinality of S to 100k. For the CO dataset, we observe that fewer top-k evaluations are necessary with increasing $|S|$. This is because the data space has more data points, thus becomes denser, and k-skyband queries have fewer weighting vectors as results, hence smaller processing cost. In Figure 9(c), we use S=10k and W=10k, and study how the value of k affects the performance of RTA. It is clear that RTA is highly efficient for UN and AC datasets, and its performance is affected only for the CO dataset. The increase of k increases the probability that a query point belongs to top-k for some weighting vector, and therefore the average cardinality of $bRTOP_k(q)$ increases, leading to more top-k evaluations.

We also study the performance of RTA for a clustered dataset W, using C_W=5 clusters of weighting vectors. A clustered dataset W simulates the case where user preferences
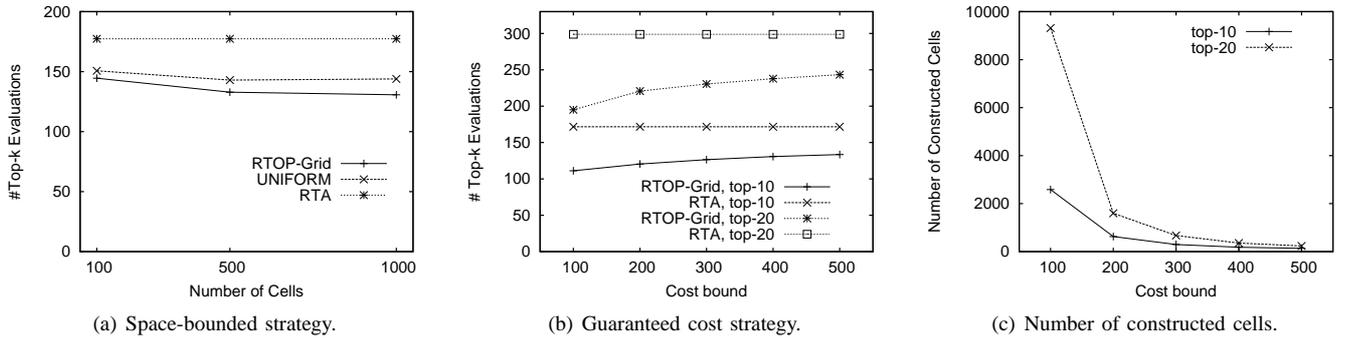
(a) Space-bounded strategy.

(b) Guaranteed cost strategy.

(c) Number of constructed cells.

Fig. 11. Performance evaluation of the strategies of RTOP-Grid.

(a) Scalability with $|W|$.

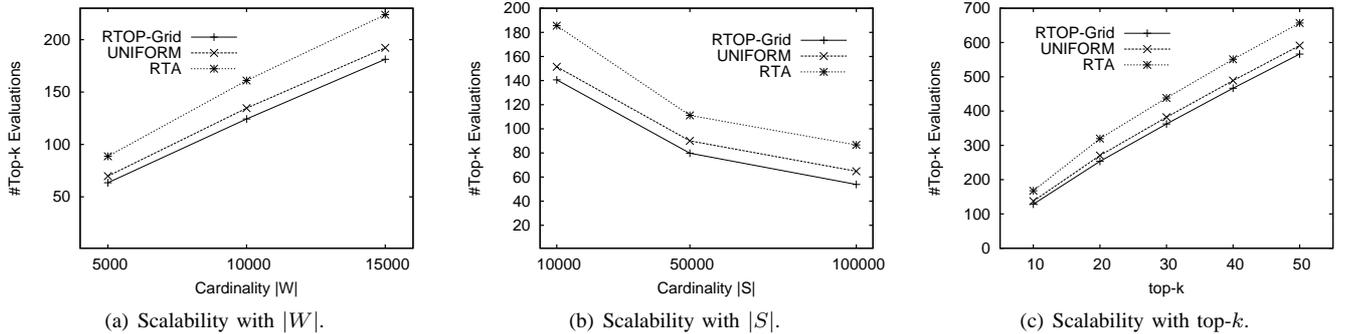(b) Scalability with $|S|$.

(c) Scalability with top-k.

Fig. 12. Scalability study of RTOP-Grid for the space-bounded strategy.

are not independent, but there exist some groups of common user preferences. This chart, depicted in Figure 10(a), is analogous (thus also comparable) to the setup of Figure 8(c), which was for a uniform dataset W. The results show that in the case of clustered dataset W, RTA performs better than for uniform W for all data distributions, nevertheless the general trends remain the same as dimensionality increases.

In Figure 10(b), we test the performance of RTA on the NBA dataset. The performance of RTA is in accordance with the case of synthetic data. We try both a uniform and clustered dataset W and the results show again that fewer top-k evaluations are required for the clustered dataset W. In Figure 10(c), a similar experiment is conducted using the HOUSE dataset.

B. Performance Evaluation of RTOP-Grid

In the sequel, we evaluate the performance of RTOP-Grid and the results are shown in Figure 11. Unless mentioned explicitly, we use $|S|$=10k, $|W|$=10k, d=5 and top-k=10. First, we provide a comparison of the RTOP-Grid space-bounded strategy to the UNIFORM approach and to RTA (Figure 11(a)), for increasing number of cells. RTOP-Grid performs consistently better than UNIFORM, demonstrating the advantages of using the cost-based splitting strategy, instead of splitting the cell with the maximum volume. RTOP-Grid also provides an improvement to RTA, in terms of the required number of top-k evaluations as expected, and in this setup it achieves a reduction of top-k evaluations between 18.5% (100 cells) and 26.3% (1000 cells).

In Figure 11(b), we test the RTOP-Grid guaranteed cost strategy versus the RTA algorithm, with increasing cost bound, for top-$k=\{10, 20\}$. The chart shows that RTOP-Grid reduces

the number of top-k evaluations compared to RTA by 30%, when the cost bound is set to 100. As expected, when the bound imposed on cost is smaller, RTOP-Grid improves RTA more. Notice that in most cases the actual number of top-k evaluations is smaller than the bound set on average cost. This is because the average cost is estimated based on the number of weighting vectors in the views, and it does not take into account the additional savings in top-k query evaluations caused by the threshold mechanism of RTA, employed also by RTOP-Grid. In Figure 11(c), we show the number of cells created by RTOP-Grid for the same experiment. Clearly, the number of cells increases rapidly when the cost bound is set too low. However, similar improvement can be obtained by relaxing the cost bound, i.e. notice that setting the bound to 200 achieves similar performance to the bound of 100, using much fewer cells. Furthermore, we study the scalability of RTOP-Grid for varying values of $|W|$, $|S|$ and top-k. Figure 12(a) shows the results obtained by increasing the cardinality of W. RTOP-Grid consistently outperforms UNIFORM and improves RTA. Then, in Figure 12(b), we set $|W|$=10k and increase $|S|$. Once again, the gains of RTOP-Grid over RTA are sustained in all setups. Finally, in Figure 12(c), the chart shows how the cost is affected by increasing values of k. RTOP-Grid performs better than RTA and UNIFORM for all k values and the benefit increases with k.

VII. RELATED WORK

Reverse top-k queries are inherently related to top-k query processing, thus we summarize some representative work here. One family of algorithms are those based on pre-processing techniques. *Onion* [2] pre-computes and stores the convex hulls of data points in layers. Then, the evaluation

of a linear top-k query is accomplished by processing the layers inwards, starting from the outmost hull. *Prefer* [6] uses materialized views of top-k result sets, according to arbitrary scoring functions. During query processing, Prefer selects the materialized view corresponding to the function that is most similar to the querying scoring function, and examines a subset of the data elements in this view. Onion and Prefer are mostly appropriate for static data, due to the high cost of pre-processing. Efficient maintenance of materialized views for top-k queries is discussed in [9]. The authors propose algorithms that reduce the storage and maintenance cost of materialized top-k views in the presence of deletions and updates. The *robust index* [8] is a sequential indexing approach that improves the performance of Onion [2] and Prefer [6]. The main idea is that a tuple should be placed at the deepest layer possible, in order to reduce the probability of accessing it at query processing time, without compromising the correctness of the result. Later, in [10], the authors propose the *dominant graph* as a structure that captures dominance relationships between points. Then, the top-k computation is mapped to a graph traversal problem. Another family of algorithms focuses on computing the top-k queries over multiple sources, where each source provides a ranking of a subset of attributes only. Fagin et al. [4] introduce two algorithms, namely TA and NRA algorithms. Variations of them have been proposed that try to improve some of their limitations and have been studied in other application areas, leading to various threshold-based algorithms [1], [3], [5], [7].

Reverse nearest neighbor (RNN) queries were originally proposed in [11] and have wide applicability in decision support systems. An RNN query finds the set of points that have the query point as their nearest neighbor. Reverse top-k queries are different from RNN queries, since the aim is to find the weights of the linear distance function that would make the query point belong to the k-nearest neighbor set of a query positioned at the origin of the data space. Recently, reverse furthest neighbor queries [14] are introduced, that are similar to RNN queries. The reverse skyline query [12] identifies customers that would be interested in a product based on the dominance of the competitors products. Monochromatic and bichromatic reverse skyline queries have been also studied in the context of uncertain databases [15]. DADA [16] aims to help manufactures to position their products in the market, based on three types of dominance relationship analysis queries. Nevertheless in these approaches, user preferences are expressed as data points that represent preferable products, whereas reverse top-k queries examine user preferences in terms of weighting vectors. Miah et al. [17] study a different problem, again from the perspective of manufacturers. They propose an algorithm that selects the subset of attributes that increases the visibility of a new product.

VIII. Conclusions

To the best of our knowledge, this is the first paper that introduces reverse top-k queries. We present two versions of reverse top-k queries, namely monochromatic and bichromatic. Then, an algorithm for evaluating monochromatic reverse top-

k queries is presented, based on the geometrical properties of the result set. Thereafter, we present an efficient threshold-based algorithm (RTA) for computing bichromatic reverse top-k queries, which eagerly discards candidate user preferences, without the need to evaluate the associated top-k query. Furthermore, we present an indexing structure based on space partitioning, which materializes reverse top-k views, in order to improve reverse top-k query processing even further. We conduct a thorough experimental evaluation that demonstrates the efficiency of our algorithms. RTA consistently improves 1 to 3 orders of magnitude the naive approach.

There are several interesting issues for future work. It is important to study in more detail the monochromatic reverse top-k query, especially for high dimensions, since the geometrical properties of the result set are essential for processing the bichromatic reverse top-k query efficiently. Moreover, we plan to study approximate reverse top-k algorithms that compute quickly a good approximation of the result set.

References

[1] R. Akbarinia, E. Pacitti, and P. Valduriez, "Best position algorithms for top-k queries," in *Proc. of VLDB*, 2007, pp. 495–506.

[2] Y.-C. Chang, L. D. Bergman, V. Castelli, C.-S. Li, M.-L. Lo, and J. R. Smith, "The onion technique: Indexing for linear optimization queries," in *Proc. of SIGMOD*, 2000, pp. 391–402.

[3] S. Chaudhuri and L. Gravano, "Evaluating top-k selection queries." in *Proc. of VLDB*, 1999, pp. 397–410.

[4] R. Fagin, A. Lotem, and M. Naor, "Optimal aggregation algorithms for middleware." in *Proc. of PODS*, 2001, pp. 102–113.

[5] U. Güntzer, W.-T. Balke, and W. Kießling, "Optimizing multi-feature queries for image databases." in *Proc. of VLDB*, 2000, pp. 419–428.

[6] V. Hristidis, N. Koudas, and Y. Papakonstantinou, "Prefer: A system for the efficient execution of multi-parametric ranked queries," in *Proc. of SIGMOD*, 2001, pp. 259–270.

[7] A. Marian, N. Bruno, and L. Gravano, "Evaluating top-k queries over web-accessible databases." *ACM Transactions on Database Systems*, vol. 29, no. 2, pp. 319–362, 2004.

[8] D. Xin, C. Chen, and J. Han, "Towards robust indexing for ranked queries," in *Proc. of VLDB*, 2006, pp. 235–246.

[9] K. Yi, H. Yu, J. Yang, G. Xia, and Y. Chen, "Efficient maintenance of materialized top-k views," in *Proc. of ICDE*, 2003, pp. 189–200.

[10] L. Zou and L. Chen, "Dominant graph: An efficient indexing structure to answer top-k queries," in *Proc. of ICDE*, 2008, pp. 536–545.

[11] F. Korn and S. Muthukrishnan, "Influence sets based on reverse nearest neighbor queries," in *Proc. of SIGMOD*, 2000, pp. 201–212.

[12] E. Dellis and B. Seeger, "Efficient computation of reverse skyline queries," in *Proc. of VLDB*, 2007, pp. 291–302.

[13] P. Tsaparas, T. Palpanas, Y. Kotidis, N. Koudas, and D. Srivastava, "Ranked join indices." in *Proc. of ICDE*, 2003, pp. 277–288.

[14] B. Yao, F. Li, and P. Kumar, "Reverse furthest neighbors in spatial databases," in *Proc. of ICDE*, 2009.

[15] X. Lian and L. Chen, "Monochromatic and bichromatic reverse skyline search over uncertain databases," in *Proc. of SIGMOD*, 2008, pp. 213–226.

[16] C. Li, B. C. Ooi, A. K. H. Tung, and S. Wang, "Dada: a data cube for dominant relationship analysis," in *Proc. of SIGMOD*, 2006, pp. 659–670.

[17] M. Miah, G. Das, V. Hristidis, and H. Mannila, "Standing out in a crowd: Selecting attributes for maximum visibility," in *Proc. of ICDE*, 2008, pp. 356–365.