Monochromatic and Bichromatic Reverse Top-k Queries

Akrivi Vlachou, Christos Doulkeridis, Yannis Kotidis, and Kjetil Nørvåg

Abstract—Nowadays, most applications return to the user a limited set of ranked results based on the individual user's preferences, which are commonly expressed through top-k queries. From the perspective of a manufacturer, it is imperative that her products appear in the highest ranked positions for many different user preferences, otherwise the product is not visible to potential customers. In this paper, we define a novel query type, namely the *reverse top-k query*, that covers this requirement: "Given a potential product, which are the user preferences that make this product belong to the top-k query result set?". Reverse top-k queries are essential for manufacturers to assess the impact of their products in the market based on the competition. We formally define reverse top-k queries and introduce two versions of the query, monochromatic and bichromatic. First, we provide a geometric interpretation of the monochromatic reverse top-k query to acquire an intuition of the solution space. Then, we study in detail the case of bichromatic reverse top-k query, and we propose two techniques for query processing, namely an efficient threshold-based algorithm and an algorithm based on materialized reverse top-k views. Our experimental evaluation demonstrates the efficiency of our techniques.

Index Terms—reverse top-k query, top-k query, user preferences

1 Introduction

Recently, the support for rank-aware query processing has attracted much attention in the database research community. Top-k queries retrieve only a ranked set of k objects that best match the user preferences, thus avoiding overwhelming result sets. Since most applications return to the user only a limited set of ranked results based on the individual user's preferences, it is imperative for a manufacturer that her products appear in the highest ranked positions for many different user preferences, otherwise the product is not visible to potential customers. In this paper, we assume that users express their preferences through linear top-k queries, which are defined by assigning a weight to each of the scoring attributes, indicating the importance of each attribute to the user. This model is in agreement with the notion of preference [1], [2] and is widely adopted in related work.

In this paper, we define a novel query type, namely the *reverse top-k query*, which can be expressed as follows: "Given a potential product, which are the user preferences for which this product is in the top-k query result set?". We formally define reverse top-k queries and study two versions of the query: monochromatic and bichromatic reverse top-k queries. In the former, there is no knowledge of user preferences and the aim is to estimate the impact of a potential product in 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.

Contributions. First, we introduce and formally define a novel query type called reverse top-k query (Section 3) and present two versions, namely monochromatic and bichromatic. We analyze the geometrical properties for the 2-dimensional case of the monochromatic reverse top-k query and provide an algorithmic solution (Section 4). Furthermore, we discuss the geometric interpretation of the solution space for higher dimensionality. Then, we study in detail the case of bichromatic reverse top-k query. 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. Instead, we propose an efficient and progressive threshold-based algorithm, called RTA, for processing bichromatic reverse top-k queries for arbitrary data dimensionality (Section 5). RTA eagerly discards candidate user preferences, without processing the respective top-k queries. In addition, we present an indexing structure based on space partitioning, which materializes reverse top-k views, in order to further improve reverse top-k query processing (Section 6). We discuss the construction, usage and maintenance of the index based on materialized reverse top-k views. We conduct a thorough experimental evaluation that demonstrates the efficiency

- *A. Vlachou, C. Doulkeridis and K. Nørvåg are with the Department of Computer Science, Norwegian University of Science and Technology, Norway.*
  E-mails: {vlachou, cdoulk, noervaag}@idi.ntnu.no
- *Y. Kotidis is with the Department of Informatics, Athens University of Economics and Business, Greece.*
  E-mail: kotidis@aueb.gr

of our algorithms (Section 7). It is noteworthy that our algorithms consistently outperform a brute force algorithm by 1 to 3 orders of magnitude in terms of required number of top-k evaluations. Finally, Section 8 reviews the related work and we conclude in Section 9.

This paper extends our preliminary work [3] and provides a more thorough study of the problem. Furthermore, we present new experimental results that lead to interesting findings and conclusions.

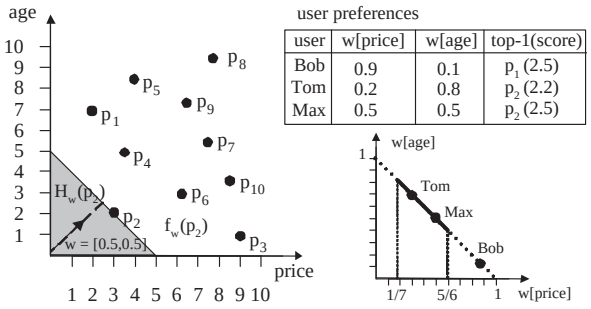


Fig. 1. Example of reverse top-k query.

2 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 $|S|$, an object $p \in S$ can be represented as a d-dimensional point $p = \{p[1], \ldots, p[d]\}$ where $p[i]$ is a value on dimension d_i. We assume that each dimension represents a numerical scoring attribute, therefore the values $p[i]$ in any dimension d_i are numerical non-negative values. Furthermore, without loss of generality, we assume that smaller scoring 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 $w[i] \geq 0$ $(1 \leq i \leq d)$, $\exists j$ such that $w[j] > 0$. The weights represent the relative importance between different dimensions, and without loss of generality we assume that $\sum_{i=1}^{d} w[i] = 1$. The weights indicate the user preferences, because they alter the ranking of the data points and therefore the top-k result set. A linear top-k query can be represented by a vector w and the result of a top-k query is a ranked list of the k points with the best scoring values f_w.

Definition 1: (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 a d-dimensional Euclidean space, there exists a one-to-one correspondence between a weighting vector w and a hyperplane ℓ which is perpendicular to w. We call the (d-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. Fig. 1 (left) 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 $f_w(p_i) = f_w(p_2) = 2.5$. Furthermore, the rank 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 $\ell_w(p)$ that contains the origin of the data space. Hence, p_2 is the top-1 result for the query $0.5 \times x + 0.5 \times y$. In the rest of the paper, we refer to this half-space as *query space of w defined by p* and denote it as $\mathcal{H}_w(p)$.

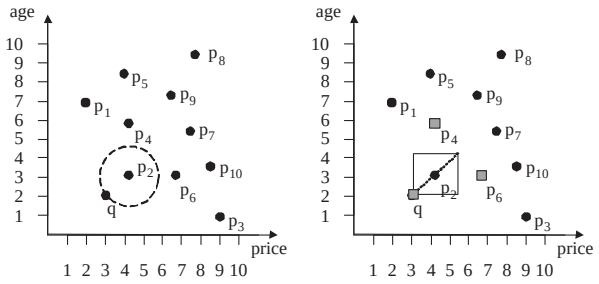
3 REVERSING TOP-k QUERIES

In this section, we introduce the concept of the reverse top-k query through an example and point out the differences to existing query types.

3.1 Example of Reverse Top-k Query

Given a database of products, a reverse top-k query returns those users (represented by weighting vectors) that rank a potential product highly. Consider for example a database containing information about different cars, depicted in Fig. 1 (left). 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 Fig. 1 (right) also depicts a set 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 (namely 2.5) for the particular weights. 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 (RTOPk) is defined by a user-specified product p and returns the weighting vectors w for which p is in the top-k result set. There exist two different versions of the reverse top-k query: the monochromatic, which does not require any knowledge of user preferences, and the bichromatic, which assumes that a dataset of preferences is given. In our example, the bichromatic reverse top-1 result set of p_1 contains the weighting vector $(0.9, 0.1)$ defined



(a) RNN vs RTOPk. (b) RSKY vs RTOPk.

Fig. 2. Differences to existing query types.

by Bob. Notice that for p_2, two weighting vectors belong to the bichromatic reverse top-1 result set $\{(0.5, 0.5), (0.2, 0.8)\}$, 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 bichromatic reverse top-1 result set of p_2. This segment of line $w[price] + w[age] = 1$ is the result set of the monochromatic reverse top-k query for the query point p=p_2.

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 estimating 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 of p_2 in the market 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 any customer based on her preferences. Summarizing, for the bichromatic version of the reverse top-k query, the result set contains a finite number of weighting vectors, while the monochromatic version identifies the partitions of the solution space that satisfy the query.

3.2 Differences to Existing Query Types

The reverse nearest neighbor query (RNN) [4] and the reverse skyline query (RSKY) [5] have been proposed for supporting decision making. In the following, we point out the differences between these query types and RTOPk queries.

Reverse top-k queries differ from reverse nearest neighbor queries [4]. Given a query point q the monochromatic RNN query retrieves all data points which have q as nearest neighbor. In the case of the car database, this is equivalent to finding all cars that are closer to the query point than to any other point of the dataset. For example, in Fig. 2(a), a RNN query returns p_2 (assuming Euclidean distance), while the result set of RTOPk query for $k = 1$ is defined by the line segment $[\frac{1}{7}, \frac{5}{6}]$ in the space of the weighting vectors. The bichromatic versions of these query types are also different. In our running example, the bichromatic RTOPk query returns all weighting vectors $w \in W$ such that $w[price] \in [\frac{1}{7}, \frac{5}{6}]$. On the other hand, the bichromatic RNN is defined based on two datasets $\mathcal{A}$, $\mathcal{B}$, and returns all points that belong to $\mathcal{A}$, that are closer to q than any point of $\mathcal{B}$. Thus, while a monochromatic/bichromatic RNN query retrieves a set of points that are closer to the query point than any other data point, the RTOPk query retrieves a set of weighting vectors or partitions of the solution space defined by the weights $w[i]$. An alternative definition of RTOPk query is, given a query point q, find the distance functions (in terms of weighting vectors) for which q belongs to the k-nearest neighbors of the point positioned at the origin of the data space.

The reverse skyline query [5], [6] has been proposed to explore the dominance relationships between products relatively to the user preferences. The user preferences are described by a data point that represents the ideal (non-existing) product for the user. In contrast, the RTOPk query assumes that the user preferences are expressed as weights that define the relative importance of each dimension. Given a query point, the RSKY query retrieves all data points for which the query point belongs to their dynamic skyline result set[1]. Thus, the RSKY query retrieves the data points that are at least in one dimension more similar (in terms of absolute difference of attribute values) to q than all other data points. For example, in Fig. 2(b), the dynamic skyline of p_2 is depicted (gray square points). Since the dynamic skyline query is defined by absolute differences, q belongs to the dynamic skyline of p_2, if and only if only p_2 and no other data point is enclosed in the depicted rectangle. As q belongs to the dynamic skyline, p_2 is in the result set of the reverse skyline of q. In the case of bichromatic reverse skyline, two datasets $\mathcal{A}$, $\mathcal{B}$ are given, each of them containing data points sharing the same attributes (in our example price and age). Then, given a query point q, the goal is to find all points belonging to $\mathcal{A}$ that are more similar in at least one dimension to q than any point of $\mathcal{B}$. Both monochromatic and bichromatic RSKY queries differ from RTOPk queries, since in the former the result set is a set of data points to which q is more similar than all other points in at least one dimension, while in the latter user preferences are returned that define linear scoring functions for which q is highly ranked.

1. A point p_i *dynamically dominates* p_i' based on point q, if $\forall d_j \in D$: $|p_i[j] - q[j]| \leq |p_i'[j] - q[j]|$, and on at least one dimension $d_j \in D$: $|p_i[j] - q[j]| < |p_i'[j] - q[j]|$.

4 MONOCHROMATIC RTOPk QUERIES

We commence by providing a formal definition of monochromatic reverse top-k for a query point q and an integer k.

Definition 2: (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 query ($mRTOP_k(q)$) of point q is the locus[2], i.e., a collection of d-dimensional vectors $\{w_i\}$, for which $\exists p \in TOP_k(w_i)$ such that $f_{w_i}(q) \leq f_{w_i}(p)$.

Let us assume that W denotes the set of all valid assignments of w. Fig. 3 shows the data and solution space of a 2-dimensional monochromatic reverse top-k query. All valid weighting vectors ($\sum_{i=1}^{d} w[i] = 1$ and $w[i] \in [0,1]$) of the reverse top-k query form the line $w[1] + w[2] = 1$ in the 2-dimensional solution space that is defined by the axis $w[1]$ and $w[2]$. Since the number of possible vectors w is infinite, it is not possible to enumerate all possible assignments of $w \in W$. 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 the query point q has the same ranking position for all weighting vectors $w \in W_i$. For the 2-dimensional case, each partition W_i is a line segment of the line $w[1] + w[2] = 1$. Then, the result set of the monochromatic reverse top-k is a set of partitions W_i of the solution space W:

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

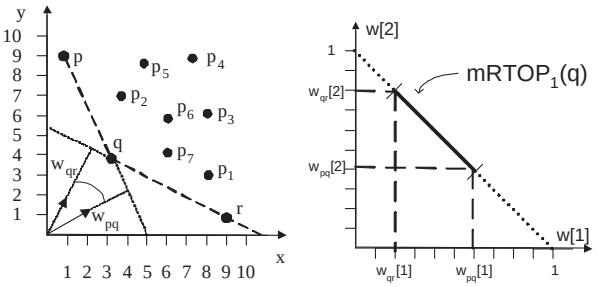
For the sake of brevity, in the rest of this paper we denote a query point $q \in TOP_k(w_i)$, instead of $\exists p \in TOP_k(w_i)$ such that $f_{w_i}(q) \leq f_{w_i}(p)$.

In this paper, we assume that any $W_i, W_j \in mRTOP_k(q)$ are non-adjacent partitions, therefore a partition W_i is the maximum partition of the solution space in which the rank of q does not change. The main topic of this section is to identify the partitions that form the result set of a monochromatic reverse top-k query. We first present an algorithm for computing the $mRTOP_k(q)$ in the 2-dimensional case. Then, we discuss the case of datasets of higher dimensionality.

4.1 Monochromatic RTOPk Query for 2d

Properties of monochromatic RTOPk query. In the following, we present some useful properties of RTOPk queries and discuss how the boundaries of the partitions W_i can be determined. We assume that the an ordering $w_1, \ldots, w_{|W|}$ of the weighting vectors of $|W|$, such that a weighting vector w_i precedes another vector w_j, if $w_i[1] < w_j[1]$. Thus, the weighting vectors w_i are ordered based on increasing angle of w_i with the y-axis.

Lemma 1: Given two points p and q such that $f_{w_1}(q) \leq f_{w_1}(p)$, there exists at most one weighting vector w such that $f_{w_i}(q) < f_{w_i}(p)$ for $w_i < w$, and $f_{w_i}(q) > f_{w_i}(p)$ for $w_i > w$.



(a) Data space (b) Solution space.

Fig. 3. Monochromatic reverse top-k query.

Based on the above lemma, the relative order of p and q changes for weighting vectors with smaller and larger angles than w. If p had a lower rank than q for vectors with smaller angle than w, then p has a higher rank for vectors with larger angle than w. If there exists such a weighting vector, then we denote it as w_{pq} and refer to it as the *weighting vector for which the relative order of q and p changes*.

Lemma 2: Given two points p and q, if there exists a weighting vector w_{pq} for which the relative order of p and q changes, then it holds that $f_{w_{pq}}(q) = f_{w_{pq}}(p)$.

Equivalently, w_{pq} is the weighting vector that is perpendicular to the line segment pq, with $w_{pq}[1] = \frac{\lambda_{pq}}{\lambda_{pq}-1}$, where $\lambda_{pq} = \frac{q[2]-p[2]}{q[1]-p[1]}$ is the slope of line segment pq. The above equation is derived by the property that $w_{pq} \perp pq$.

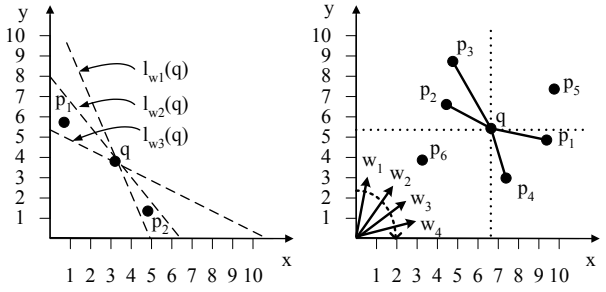
The boundaries of any partition W_i are defined by weighting vectors w_{pq} for which the relative order of q and points $p \in S$ changes (additionally, the first and last partition are defined by the weighting vectors $[0,1]$ and $[1,0]$ respectively). Intuitively, as long as the relative order between any two points does not change, the top-k result is not affected and thus the rank of q remains the same.

Lemma 3: There exists at most one partition W_i, such that for all the weighting vectors $w \in W_i$ it holds that $q \in TOP_1(w)$.

Since the relative order between q and any data point p changes only once, if the rank of p becomes higher than q, then it cannot change again for the next vectors. Thus, q cannot be in the top-1 result set for any $w > w_{pq}$. Therefore, the result set $mRTOP_1(q)$ contains at most one partition W_i of W.

Example of $mRTOP_k(q)$ for k=1. Consider for example the dataset depicted in Fig. 3(a). Since the only points that belong to the convex hull [7] are p, q and r, we conclude that (1) only these points belong to the top-1 result set for any weighting vector, and (2) there exists at least one weighting vector w_i for which $q \in TOP_1(w_i)$, and based on Lemma 3 exactly one partition $W_i \in mRTOP_1(q)$. The boundaries of the partition W_i are defined by the weighting vectors

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



(a) Multiple partitions. (b) $mRTOP_k(q)$ algorithm.

Fig. 4. Examples of $mRTOP_k(q)$ queries.

w_{pq}, w_{qr} for which the relative order between q and p or r changes. All weighting vectors w for which the following inequality holds are in the reverse top-1 result set of q: $w_{qr}[1] \leq w[1] \leq w_{pq}[1]$. The result set of $mRTOP_1(q)$ is a segment (partition) of the line $w[1] + w[2] = 1$ in the 2-dimensional solution space defined by w_{pq} and w_{qr}, as shown in Fig. 3(b).

Even though the result set $mRTOP_k$ for $k = 1$ contains at most one partition, for a reverse top-k query with $k > 1$, the result set may contain more than one partitions W_i. Consider for example the three data points in Fig. 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 reverse top-k query would return two partitions W_i.

Monochromatic RTOPk algorithm. Algorithm 1 describes the monochromatic reverse top-k algorithm for the 2-dimensional case. Data points that are dominated[3] 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 Fig. 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 vectors. Thus, our algorithm examines[4] only such incomparable points $\{p_i\}$ to q (line 5), because they alter the rank of q. The boundaries of the partitions of $mRTOP_k$ are defined by a subset of the weighting vectors $w_i = w_{p_iq}$, therefore we keep them in list W' (line 7). Then, we identify the partitions for which q belongs to the top-k, by processing W', as explained in the following example.

In Fig. 4(b), after the sorting by increasing value of $w_i[1]$ (line 10) the set W' is $\{w_1, w_2, w_3, w_4\}$ corresponding to the lines p_1q, p_2q, p_3q, p_4q 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 $H_{w_0}(q)$ (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, then 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. Then, the next partition is $W_1 = [w_1, w_2]$. Since $p_1 \in R$ (line 18), this means that the relative order of the points p_1 and q changes in W_1, 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. Notice that adjacent partitions can be easily detected and merged into one partition.

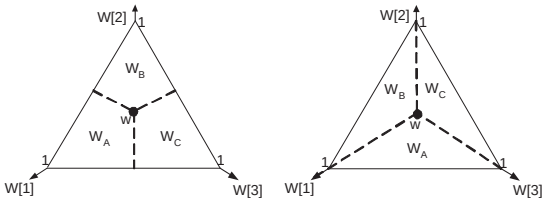
Algorithm 1 Monochromatic RTOPk Algorithm.

```
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 increasing value of 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
```

Given a query point q, let $I \subset S$ be the set of incomparable points to q. Then, Algorithm 1 produces at most $|I| + 1$ partitions. Since adjacent partitions are merged, in worst case every second partition will be in the result set of $mRTOP_k(q)$. Thus, the maximum number of partitions in $mRTOP_k(q)$ is $\left\lceil \frac{|I|+1}{2} \right\rceil$. If all data points are incomparable to q then $|I| = |S|$, which leads to an upper bound for the number of partitions. Notice that the expected number of incomparable points is much smaller. For example, assuming uniform data distribution and given that $0 \leq p[i] \leq 1, \forall p \in S$ and $1 \leq i \leq 2$, the expected number of incomparable points $|I|$ is equal to the aggregate area of the upper left quadrant and lower right quadrant defined by q multiplied with $|S|$:

3. A point p dominates q ($p \prec q$) , if $\forall d_i \in D$, $p[i] \leq q[i]$; and on at least one dimension $d_j \in D$, $p[j] < q[j]$.

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



(a) Partitioning for S_1. (b) Partitioning for S_2.

Fig. 5. Solution space for 3-dimensional data.

$$|I| = |S| - |S| \cdot q[1] \cdot q[2] - |S| \cdot (1 - q[1]) \cdot (1 - q[2])$$
$$= |S| \cdot (q[1] + q[2] - 2 \cdot q[1] \cdot q[2])$$

4.2 Higher Dimensional Data

In higher dimensions (d>2), all valid weighting vectors of the RTOPk query form a (d-1)-dimensional hyperplane that contains the points $w_i[j]$=0 $\forall j \neq i$ and $w_i[j]$=1 for j=i and $1 \leq i \leq d$. A monochromatic RTOPk query returns the partitions W_i of the hyperplane, for which the query point q is in the $TOP_k(w), \forall w \in W_i$. In the following, we provide an example for finding the partitions for d>2.

Let us consider a 3-dimensional dataset S_1 containing only three points A=$[1,0,0]$, B=$[0,1,0]$ and C=$[0,0,1]$. We denote as W_A, W_B and W_C the partitions for which A, B and C are the top-1 data point respectively. Similar to the 2-dimensional case, the borders of the partitions are defined by the weighting vectors for which the relative order between two points changes. To define the borders of the partitions W_A and W_B, we need to examine the locus of weights w' for which $f_{w'}(A)$=$f_{w'}(B)$. From this equation, we derive that w'=$[\frac{1-c}{2}, \frac{1-c}{2}, c]$ where $c \in [0,1]$. The vectors w' form a line that divides the solution space into two partitions.

Furthermore, we seek only the weighting vectors for which $f_{w'}(A)$=$f_{w'}(B)$<$f_{w'}(C)$, since otherwise C is the top-1 point. Thus, if we take also into consideration that $f_{w'}(A) < f_{w'}(C)$, then an additional constraint is formed, namely $c > \frac{1}{3}$. Therefore, the border between the partitions W_A and W_B is the line segment defined by the points $[1/3, 1/3, 1/3]$ and $[1/2, 1/2, 0]$. By repeating the same procedure for the other pairs of points, the result is the partitioning depicted in Fig. 5(a). Notice that there exists a single weighting vector w=$[1/3, 1/3, 1/3]$ for which all three data points have the same score for the dataset S_1.

Fig. 5(b) depicts the partitions of the solution space for another dataset S_2 containing the points A=$[1/2, 0, 1/2]$, B=$[1/2, 1/2, 0]$ and C=$[0, 1/2, 1/2]$. For dataset S_2, in order to detect the border between W_A and W_B, only the constraint changes to $c < \frac{1}{3}$, therefore the border between the partitions W_A and W_B is defined as the line segment defined by the points $[1/3, 1/3, 1/3]$ and $[1, 0, 0]$. This discussion shows that obtaining the partition boundaries in higher dimensions is a complicated process that goes far beyond the task of identifying the weighting vectors for which the relative order changes.

Algorithm 1 can be extended for higher dimensions, similarly to the approach in [2] for traditional top-k query processing. The main difference is that in higher dimensions, in each repetition (line 4), each pair of points define a (d-2)-dimensional hyperplane in the solution space. The remaining challenge is to define the boundaries of the partitions. The details of such a generalization are very interesting and we plan to study them further in our future work.

Algorithm 2 RTA: RTOPk Threshold Algorithm.

```
1: Input: S, W, q, k
2: Output: bRTOP_k(q)
3: W' ← ∅, buffer ← ∅
4: τ ← ∞
5: for (each w_i ∈ W) do
6:    if (f_{w_i}(q) ≤ τ) then
7:       buffer ← TOP_k(w_i)
8:       if (f_{w_i}(q) ≤ τ_{w_i}(buffer)) then
9:          W' ← W' ∪ {w_i}
10:      end if
11:   end if
12:   τ ← τ_{w_{i+1}}(buffer)
13: end for
14: return W'
```

5 Bichromatic RTOPk Queries

Definition 3: (Bichromatic Reverse top-k) Given a point q and a positive number k, as well as two datasets S and W, where S represents data points and W is a dataset containing different weighting vectors, a weighting vector $w_i \in W$ belongs to the bichromatic reverse top-k result set ($bRTOP_k(q)$) 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 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 test whether q belongs to $TOP_k(w_i)$. Obviously, the brute force approach is prohibitively expensive and does not scale with the number of weighting vectors w_i in the dataset W which may be high (comparable to the size $|S|$ of the dataset S). In the sequel, we present a threshold-based algorithm, called RTA (**R**everse top-k **T**hreshold **A**lgorithm), which discards weighting vectors that cannot contribute to the result set $bRTOP_k(q)$, without evaluating the corresponding top-k queries.

5.1 Threshold-based Algorithm (RTA)

RTA aims to reduce the number of top-k query evaluations, based on the observation that top-k queries

defined by similar weighting vectors[5] return similar result sets [1]. Hence, RTA exploits already computed top-k result sets to avoid evaluating weighting vectors that cannot be in the reverse top-k result set. Therefore, in each repetition a threshold is set based on the previously computed top-k result set P. Given a set of points P, we denote as $\tau_{w_i}(P) \equiv max\{f_{w_i}(P)\}$ the maximum of all scoring values $f_{w_i}(p_j)$, $p_j \in P$, which means that $\forall p_j \in P$: $\tau_{w_i}(P) \geq f_{w_i}(p_j)$, and $\exists p_j \in P$: $\tau_{w_i}(P) = f_{w_i}(p_j)$. The maximum value $\tau_{w_i}(P)$ corresponds to the worst scoring value for any point in the set P based on w_i and is used as a threshold during the reverse top-k evaluation.

Algorithm 2 formally describes the RTA algorithm for processing a bichromatic RTOPk 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 τ 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. The score $f_{w_i}(q)$ of query point q based on vector w_i is computed and compared against $\tau_{w_i}(\textit{buffer})$ (line 8) and if it is not greater than $\tau_{w_i}(\textit{buffer})$, then w_i is added to the result set (line 9). Before the next iteration, we take the next weighting vector (w_{i+1}) and we set the threshold τ equal to $\tau_{w_{i+1}}(\textit{buffer})$ (line 12). Then, the condition of line 6 is tested, and if the score $f_{w_i}(q)$ is higher than the threshold τ, then w_i can be safely discarded. If w_i cannot be discarded, we pose again a top-k query on dataset S and we update the 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. The algorithm terminates when all weighting vectors have been evaluated or discarded. Notice that the size of the buffer is always bound by k, and queries with small k values are commonly used in practice. Furthermore, we assume that the buffer contains k points, which always holds if $k < |S|$.

Theorem 1: (Correctness of the algorithm) RTA always returns the correct and the complete result set.

Proof: Equivalently, the theorem states that RTA reports a weighting vector w as result, if and only if it belongs to the result set $bRTOP_k(q)$. We prove – by contradiction – that w is never falsely reported as result and that w is never falsely discarded.

(1) Let w be a weighting vector that is falsely added to the $bRTOP_k(q)$ set, i.e., $w \notin bRTOP_k(q)$ and $f_w(q) \leq \tau_w(\textit{buffer})$. Based on line 7 $f_w(q) \leq \tau_w(TOP_k(w))$. Thus, $\exists p \in TOP_k(w)$ such that $f_w(q) \leq \tau_w(TOP_k(w)) = f_w(p)$. Then, by definition $w \in bRTOP_k(q)$, which is a contradiction.

(2) Let $w \in bRTOP_k(q)$ be a weighting vector that is falsely discarded. Then, based on the definition of the reverse top-k query, $\exists p \in TOP_k(w)$ such that $f_w(q) \leq f_w(p)$. Since RTA discarded w, there exists a set of k points p_i ($1 \leq i \leq k$) in the buffer that have a better scoring value than q based on the threshold, i.e., $\forall p_i$: $f_w(p_i) < f_w(q) \leq f_w(p)$. This means that $p \notin TOP_k(w)$, which leads to a contradiction. □

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

5.2 Sorted Access to Weighting Vectors

In each repetition, RTA sets a threshold exploiting previously computed top-k result sets, in order to discard weighting vectors that cannot be in the query result set. The effectiveness of the threshold depends on the k data objects in the buffer. Top-k queries defined by similar weighting vectors return similar result sets [1]. Thus, if the buffered result set was obtained by a similar weighting vector w' with the currently processed w, then the probability that the threshold can discard w is high. As a result, the order in which the weighting vectors are examined influences the performance of RTA, and it is beneficial to access similar weighting vectors in consecutive steps. Consequently, the weighting vectors W are sorted based on their pairwise similarity.

Given a similarity function $sim(w_i, w_j)$ between w_i and w_j, and an ordering of the weighting vectors $e = w_1, ..., w_{|W|}$, the overall similarity is defined as $sim(e) = \sum_{i=1}^{|W|-1} sim(w_i, w_{i+1})$. The goal is to find the optimal ordering $\hat{e}$ in terms of similarity, defined as $\hat{e} = argmax_{\forall e}(sim(e))$, that maximizes the similarity of all consecutive pairs of weighting vectors. In the following, we define the Vector Ordering Problem (VOP) formally.

Definition 4: (Vector Ordering Problem) Given a real number c and a set of vectors W with nonnegative cost function $sim(w_i, w_j)$ associated with each pair of vectors w_i and w_j, the problem is whether there exists an ordering of the weighting vectors $e = w_1, ..., w_{|W|}$, such that $\sum_{i=1}^{|W|-1} sim(w_i, w_{i+1}) \geq c$.

Lemma 4: The VOP problem is NP-complete.

Proof: We first show that VOP belongs to NP. Given an instance of VOP and a candidate solution, the verification algorithm checks that the ordering contains each vector exactly once, sums up the cost values, and checks whether the sum is at least c. This process can be done in polynomial time. To prove that VOP is NP-complete, we show that the *Traveling*

5. We address the issue of accessing similar weighting vectors in Section 5.2.

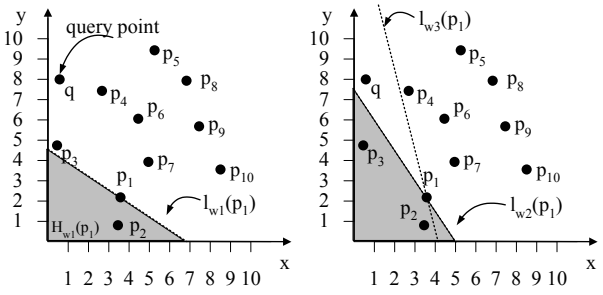
Salesman Problem[6] (TSP) is polynomial-time reducible to the sorting problem (TSP $\leq_P$ sorting problem). Let $\langle G(V,E), d, c\rangle$ be an instance of TSP. We construct an instance of VOP as follows. We form the set of vectors W by adding a vector w_i for each $v_i \in V$ and define the cost function sim as $sim(w_i, w_j) = sim(w_j, w_i) = d(v_i, v_j)$. Then, the instance of VOP is $\langle W, sim, c\rangle$, which can be created easily in polynomial time. We now show that there exists a Hamiltonian cycle with at least cost c for TSP, if and only if there exists an ordering with at least cost c for VOP. Suppose that graph G is a Hamiltonian cycle represented by the sequence $v_1, v_2, \ldots, v_{|V|}, v_1$ with at least cost c, then the vector ordering $w_1, w_2, \ldots, w_{|V|}$ has at least cost c, since $sim(w_i, w_j) = d(v_i, v_j)$. Conversely, suppose that a vector ordering $w_1, w_2, \ldots, w_{|W|}$ has at least cost c. Then, the cycle represented by the sequence $v_1, v_2, \ldots, v_{|V|}, v_1$ is a Hamiltonian cycle and has at least cost c. Thus, we conclude that VOP is NP-complete. □

The problem of finding the optimal ordering $\hat{e}$ in terms of similarity is the maximization problem of VOP. Since VOP is NP-complete, the optimization problem is NP-hard. Any algorithm proposed for solving the TSP problem can be used for finding an approximate solution of our optimization problem, if we consider a fully connected graph where each weighting vector corresponds to a vertex and the weights of the edges correspond to the similarity of the weighting vectors. More accurately, the algorithm must solve the non-metric TSP problem, since the weights on graph edges do not necessarily satisfy the triangle inequality. We employ a greedy algorithm that is known as nearest neighbor algorithm for TSP [8] to obtain an ordering of the set W with low computational overhead. We select as first weighting vector w_1 the most similar vector to the diagonal vector of the space. Then, each time, the most similar weighting vector w_{i+1} to the previous vector w_i is selected.

We employ the cosine similarity as the similarity function $sim(w_i, w_j)$ between two vectors. Notice that this sorting of the weighting vectors takes place in a preprocessing phase, since it is independent of the query point. Thus, W is stored sorted and it is given as input to RTA.

5.3 RTA Example

Consider a dataset S consisting of the points p_i, a dataset $W = \{w_1, w_2, w_3\}$ with $w_1 = [0.4, 0.6]$, $w_2 = [0.6, 0.4]$, and $w_3 = [0.8, 0.2]$, as well as the query point q, depicted in Fig. 6. Let us assume that k=2 and the first examined weighting vector is w_1. Then, RTA evaluates a top-2 query ($TOP_2(w_1)$) and retrieves the data points p_1 and p_2 that are placed in the buffer $\{p_2, p_1\}$. As depicted in Fig. 6(a), points p_1 and p_2 are enclosed in the query space $\mathcal{H}_{w_1}(p_1)$ (depicted as gray triangle). Since q is not enclosed in $\mathcal{H}_{w_1}(p_1)$, at least two data points have a better score than q and w_1 does not belong to the $bRTOP_2(q)$. This is detected by RTA in line 6, where the scoring value $f_{w_i}(q)$ is compared to the threshold.



(a) Query evaluation of w_1. (b) Processing w_2 and w_3.

Fig. 6. Example of bichromatic algorithm (RTA).

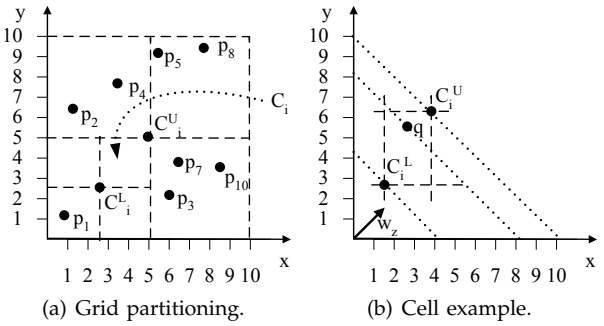
In the next step (Fig. 6(b)), w_2 is examined and the threshold is set based on the query line $l_{w_2}(p_1)$. Notice that the threshold is set equal to the maximum scoring value of points p_1 and p_2 in the buffer. Since q has a higher scoring value $f_{w_2}(q)$ than the threshold, the weighting vector w_2 is discarded without further processing. As depicted in Fig. 6(b), $\mathcal{H}_{w_2}(p_1)$ contains at least 2 data points (in this example: $\{p_1, p_2, p_3\}$), and this verifies that w_2 can be safely discarded.

When the next vector w_3 is considered, the threshold is set based on point p_1, which has the highest score for w_3 among the data points in the buffer. Then, 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 $\{p_3, p_2\}$. The score value of q is better than the score value of p_2, which is the top-2 data point for this query, so w_3 is added to the reverse top-2 result set of q. Then, RTA terminates and returns w_3 as the result of $bRTOP_2(q)$.

5.4 Incremental Threshold Refinement

In principle, RTA is independent of the algorithm used for the underlying top-k evaluation. Nevertheless, if the top-k algorithm is incremental (as in the case of the branch-and-bound algorithm that uses an R-tree), then RTA can be adapted, so that the threshold is refined after each retrieved data object. Instead of retrieving the entire top-k result set and updating the buffer afterwards (line 7), RTA may retrieve incrementally the k data objects. Each time a data object is retrieved, it is added to the buffer by keeping the k objects with the lowest scores. The threshold is updated and RTA tests if the weighting vector w_i can be discarded, before retrieving the next result. Therefore, fewer than k retrieved data objects may suffice to discard w_i.

6. Given a complete graph $G = (V, E)$, a cost function $d(v_i, v_j)$, and a nonnegative integer c, check whether G has a Hamiltonian cycle with cost at least c.



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

Fig. 7. Example of grid-based algorithm.

Algorithm 3 GRTA: Grid-based RTOPk Algorithm.

```
1: Input: S, q, k
2: Output: bRTOP_k(q)
3: W' ← ∅, W'' ← ∅, W_cand ← ∅
4: Find cell C_i that encloses q
5: for (∀w_z ∈ L_i^L) do
6:    if (w_z ∈ L_i^U) then
7:       W' ← W' ∪ {w_z}
8:    else
9:       W_cand ← W_cand ∪ {w_z}
10:   end if
11: end for
12: W'' ← RTA(S,W_cand,q,k)
13: return {W' ∪ W''}
```

The incremental refinement of the threshold may require fewer than k retrieved data objects per top-k evaluation to discard a weighting vector. On the other hand, when this occurs, fewer than k points in the buffer are updated, and the threshold for the next weighting vector is less accurate. This may cause a top-k evaluation for the next weighting vector, that could be avoided if all the k elements were retrieved. We study this effect in our experimental evaluation.

6 Materialized RTOPk 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 RTOPk queries. First, we define the indexing structure and present the properties of RTOP-Grid that improve the performance of RTOPk queries. Afterwards, we present the RTOPk algorithm that uses the RTOP-Grid and the construction algorithm of RTOP-Grid that takes into account the gain in computational cost during query processing. Finally, we generalize our approach for arbitrary k values and discuss updates.

6.1 Definitions and Properties

Let us assume a grid-based space partitioning of the data space. The grid consists of disjoint data space partitions, also called *cells* (Fig. 7(a)). Each cell C_i is defined by its *lower left corner* C_i^L and *upper right corner* C_i^U. Given a cell C_i and a 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, 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 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. Henceforth, we refer to the lists of weighting vectors of a cell as materialized views.

During query processing we exploit the materialized views of the cells, in order to restrict the number of candidate weighting vectors that need to be examined by RTA algorithm. Given a query point q, the cell C_i which encloses query point q is determined. Based on the following theorem the materialized views can be used for restricting the computational cost of the RTOPk query.

Theorem 2: Given a query point q and a cell C_i that encloses q, it holds that:

(1) If a weighting vector $w \in W$ does not exist in the materialized view $w \notin L_i^L$, then w cannot be in the reverse top-k result set of q: $w \notin bRTOP_k(q)$.

(2) If a weighting vector $w \in W$ belongs to the materialized view $w \in L_i^U$, then w is in the reverse top-k result set of q: $w \in bRTOP_k(q)$.

Proof: It holds that $C_i^L[i] \leq q[i] \leq C_i^U[i]$ for $1 \leq i \leq d$. Thus, for any $w \in W$ it holds that $f_w(C_i^L) \leq f_w(q) \leq f_w(C_i^U)$ due to the monotonicity of f_w. Therefore, if $w \notin L_i^L$ then $w \notin bRTOP_k(q)$, whereas if $w \in L_i^U$ then $w \in bRTOP_k(q)$. □

As an example, in Fig. 7(b), for any weighting vector w_z, the query space $\mathcal{H}_{w_z}(C_i^U)$ contains the query space $\mathcal{H}_{w_z}(q)$, which in turn contains the query space $\mathcal{H}_{w_z}(C_i^L)$. If $w_z \notin L_i^L$, then the query space $\mathcal{H}_{w_z}(C_i^L)$ contains more than k data points, which means that $\mathcal{H}_{w_z}(q)$ contains also more than k data points ($q \notin TOP_k(w_z)$). On the other hand, if $w_z \in L_i^U$ then fewer 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 w_z that are in L_i^L but not in L_i^U we need to examine if q belongs to the $TOP_k(w_z)$ result set. Essentially, this restricts the input of Algorithm 2 to weighting vectors only from the set $L_i^L - L_i^U$, rather than W.

6.2 Grid-based RTOPk Algorithm (GRTA)

Algorithm 3 formally describes the evaluation of an RTOPk query 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 based on Theorem 2 we are certain that w_z belongs to the $bRTOP_k(q)$ result set and w_z is added to list W' (line 7). 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).

An important improvement of the grid-based materialization compared to the RTA algorithm is that some weighting vectors are added to the result set without evaluating the top-k query. Furthermore, the number of weighting vectors that need to be examined in order to retrieve the RTOPk result set 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 $|L_i^L| - |L_i^U|$. Of course, RTA reduces even more this number, by discarding weighting vectors based on already computed results. 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.

6.3 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 RTOPk 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 *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:

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

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: $L_0^L \leftarrow$ RTA(S,W,C_0^L,k)
5: $L_0^U \leftarrow$ 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: $L_1^U \leftarrow$ GRTA(S,C_1^U,k)
13: $L_2^L \leftarrow$ 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

In the case of uniform query distribution, the integral of Equation 1 can be replaced by the fraction of the volume of the space D 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:

$$COST_{RTOP-Grid} = \sum_{\forall i} COST_{C_i} \qquad (2)$$

The cost function implies 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 and picks in each iteration 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 RTOPk 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. Secondly, 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. Then, 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 condition of Algorithm 4 (line 8) is modified as follows: $COST_{RTOP-Grid}$>$Limit$.

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.

6.4 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}
```

6.5 Updates

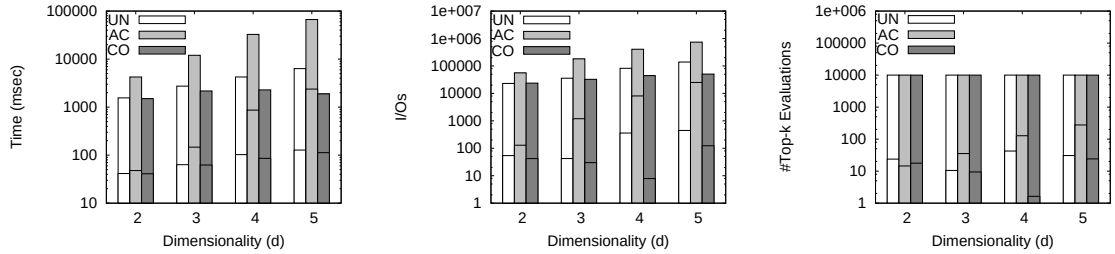
Updates that occur either in W or S affect the materialized RTOPk views, therefore they should be supported efficiently. In case of insertion of a new weighting vector w_{ins}, we need to progressively examine the corners of the grid, starting from the origin of the data space. If a corner C_i^L (C_i^U) 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.

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 non-dominating cell corners, therefore we need to recompute the materialized views for them, since new weighting vectors may have to be added.

7 Experimental Evaluation

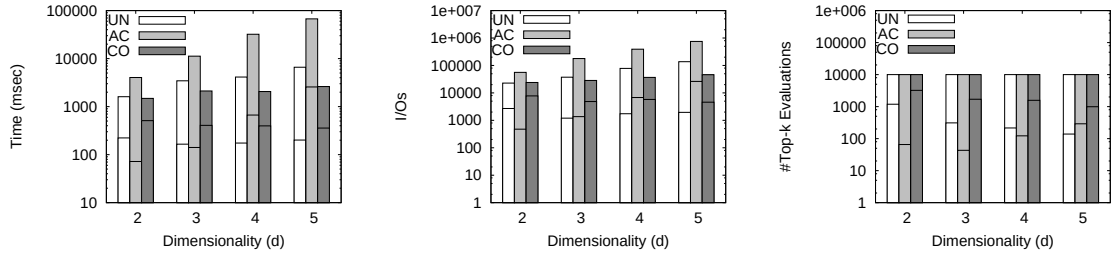
In this section, we present an extensive experimental evaluation of reverse top-k queries. All algorithms are implemented in Java and the experiments run on a 3GHz Dual Core AMD processor equipped with 2GB RAM. The block size is 8KB.

As far as the dataset S is concerned, both real 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 correlated and anticorrelated datasets are generated as described in [9]. We also use two real datasets: NBA (17265 tuples, d=5) and HOUSE (127930 tuples, d=6) [3]. For the dataset W, two different data distributions are examined, namely uniform (UN) and clustered (CL). For the clustered dataset W, first C_W



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

Fig. 8. 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. 9. Performance of RTA for varying d for k-skyband queries [naive (outer bar) vs. RTA (inner bar)].

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 evaluate the performance of RTA against an alternative technique (referred as naive) that evaluates a top-k query for each weight in the dataset W. In particular, both for RTA and naive, the dataset S is indexed by an R-tree and top-k processing is performed using a state-of-the-art branch-and-bound algorithm. Our metrics include: a) the time (wall-clock time), b) the I/Os, and c) the number of top-k evaluations required by each algorithm. 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 cost is the same for every method.

7.1 Performance Evaluation of RTA

In Fig. 8, 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 bar depicts the performance of RTA. Regarding 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 Fig. 8(c), where the number of top-k evaluations necessary for computing an RTOPk 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 evaluations in all cases.

An interesting observation is that only a small percentage (around 2%) of the queries actually return non-empty result sets. Since the queries are generated following the data distribution, many queries are not in the top-k result for any weighting vector. Reverse top-k queries with empty result sets are also very informative for a product manufacturer, since they indicate that the particular product is not popular for any customer, compared to their competitors' products. On the other hand, RTA processes RTOPk queries that have a small or empty result set efficiently by often requiring only one top-k evaluation, because the threshold employed eliminates candidate weighting vectors that do not belong to the result set. In contrast, naive does not have this ability and always computes $|W|$ top-k queries. In order to generate a more challenging query workload for RTA, we increase the probability that a query point belongs to a top-k result, by selecting random query points from the k-skyband[7] of the dataset. Obviously, these query points

7. A k-skyband query returns the set of points which are dominated by at most k-1 other points.

are more likely to produce non-empty reverse top-k results. This query workload corresponds to queries about products that seem popular to customers, and manufacturers are expected to pose such queries with high probability.

Fig. 9 depicts the results obtained by using k-skyband queries for the same experimental setup depicted in Fig. 8. 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 Fig. 9(c). First, we notice that the correlated dataset requires more top-k evaluations. The reason is that the k-skyband of a correlated dataset contains few 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 in Fig. 10. 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 using different data distributions of S (Fig. 10(a)). We fix the remaining parameters to $|S|$=10K, d=5 and top-k=10. 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). Thus, RTA needs only 544 (10.88%) out of 5000 query evaluations (100%), which is just marginally more than the mandatory 459 (9.18%), and therefore RTA saves 89.12% of the cost.

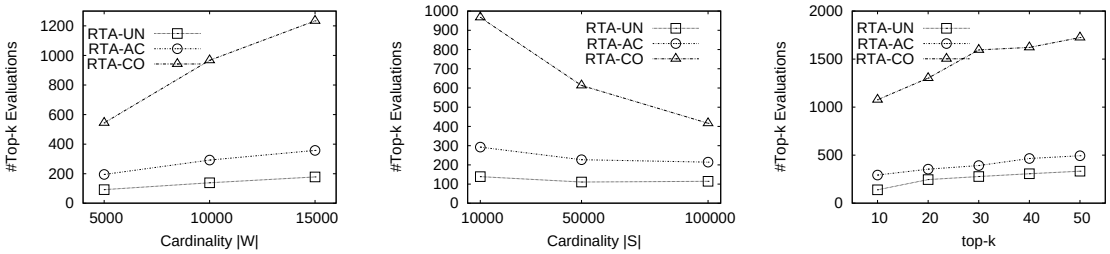
In Fig. 10(b), we set $|W|$=10K and gradually increase the cardinality of S to 100K. For the CO dataset, we observe that fewer top-k evaluations are necessary with increasing $|S|$. The main reason is that the data space has more data points, thus becomes denser, and k-skyband queries lead to result sets with fewer weighting vectors, hence smaller processing cost. In Fig. 10(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 CO. Higher values of k increase 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 are not independent, but there exist some groups of common user preferences. In this experiment, we use the default setup and vary the dimensionality. Thus, Fig. 11(a) is analogous and also comparable to Fig. 9(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 Fig. 11(b), we assess RTA using 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 Fig. 11(c), a similar experiment is conducted using the HOUSE dataset.

7.2 Performance Evaluation of RTOP-Grid

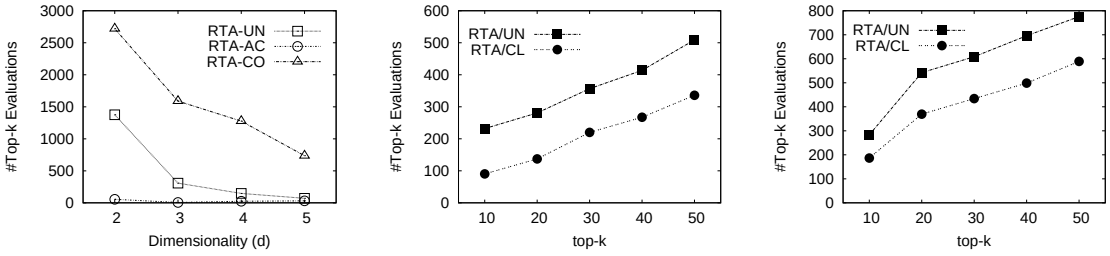
In the sequel, we evaluate the performance of RTOP-Grid in Fig. 12. Unless mentioned explicitly, we use $|S|$=10K, $|W|$=10K, d=5 and top-k=10. First, we provide a comparison of the RTOP-Grid space-bounded strategy to the UNIFORM approach and to RTA (Fig. 12(a)), for increasing number of cells. RTOP-Grid 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 Fig. 12(b), we test the RTOP-Grid guaranteed cost strategy versus RTA, 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 Fig. 12(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 improvements 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. Fig. 13(a) shows the results obtained by increasing $|W|$. RTOP-Grid consistently outperforms UNIFORM and improves RTA. Then, in



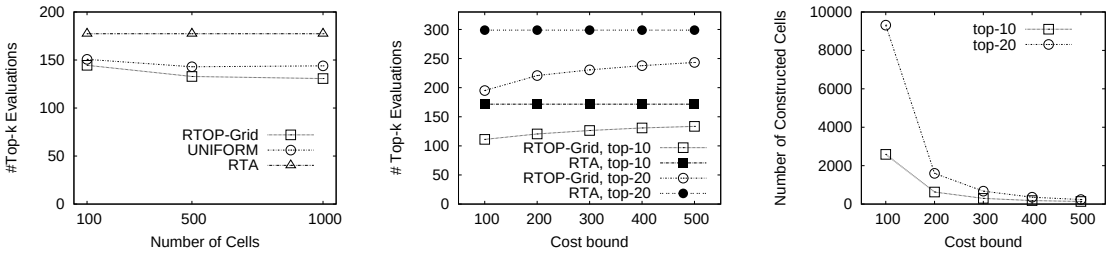
(a) Scalability with $|W|$. (b) Scalability with $|S|$. (c) Scalability with top-k.

Fig. 10. Scalability study of RTA.



(a) Clustered dataset W. (b) Real dataset (NBA). (c) Real dataset (HOUSE).

Fig. 11. Performance of RTA for clustered weights W and for real data (NBA and HOUSE).



(a) Space-bounded strategy. (b) Guaranteed cost strategy. (c) Number of constructed cells.

Fig. 12. Performance evaluation of the strategies of RTOP-Grid.

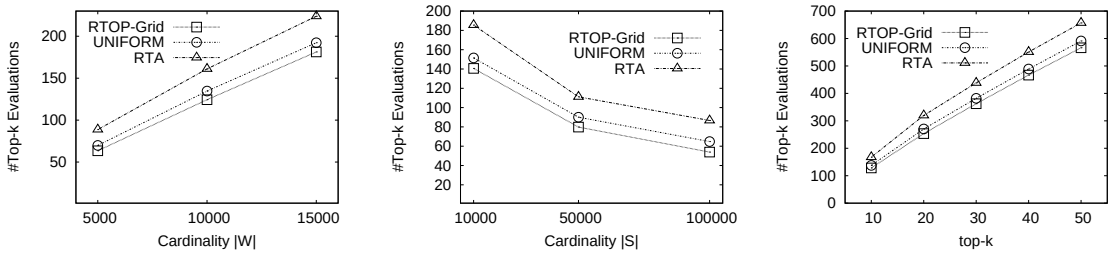
Fig. 13(b), we set $|W|$=10K and increase $|S|$. Once again, the gain of RTOP-Grid over RTA is sustained in all setups. Finally, in Fig. 13(c), the chart shows how the cost is affected by increasing k. RTOP-Grid performs better than RTA and UNIFORM for all k values and the benefit increases with k.

7.3 Monochromatic vs. Bichromatic Algorithm

In Fig. 14, we set d=2 and study the comparative performance of RTA against the monochromatic RTOPk algorithm. To this end, we apply the monochromatic algorithm to identify the partitions of W that belong to the solution set, and then retrieve the subset of weighting vectors W that belong to these partitions. The data distribution for both S and W is uniform, denoted as UN/UN. The default values used are $|S|$=10K, $|W|$=10K, and top-k=10.

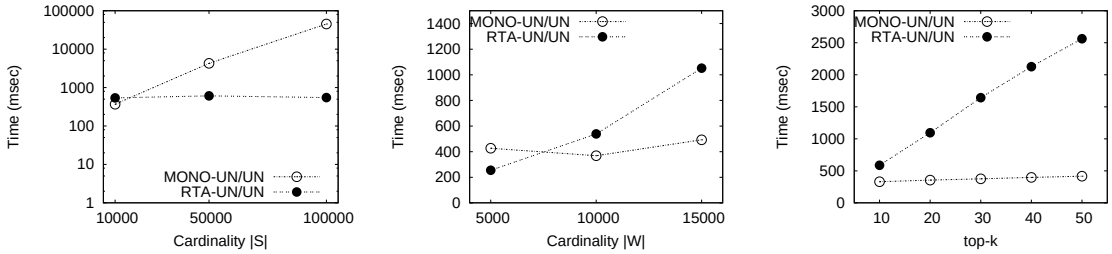
In Fig. 14(a), we measure the time required by each algorithm employing a logarithmic scale. We observe that RTA scales better with $|S|$, compared to the monochromatic algorithm. In Fig. 14(b), we vary the cardinality of W, from 5K to 15K vectors. It turns out that the monochromatic reverse top-k algorithm scales better with $|W|$, because it is immune to the actual cardinality of W, as it only needs to identify the partitions of the space that provide solutions to the query. The number of weighting vectors does not affect significantly the time required to identify the partitions. However, RTA needs to examine (and potentially process) more top-k queries as the cardinality of W increases, therefore its total time increases. Similarly, in Fig. 14(c), RTA requires more time to compute the result for increasing values of top-k. Again, the monochromatic reverse top-k algorithm is practically unaffected by the increased values of k.

All in all, RTA is more sensitive to higher values of $|W|$ and top-k, thus the monochromatic RTOPk algorithm scales better. In contrast, the monochromatic algorithm is sensitive to the cardinality of S, therefore RTA performs better for increased values of $|S|$.



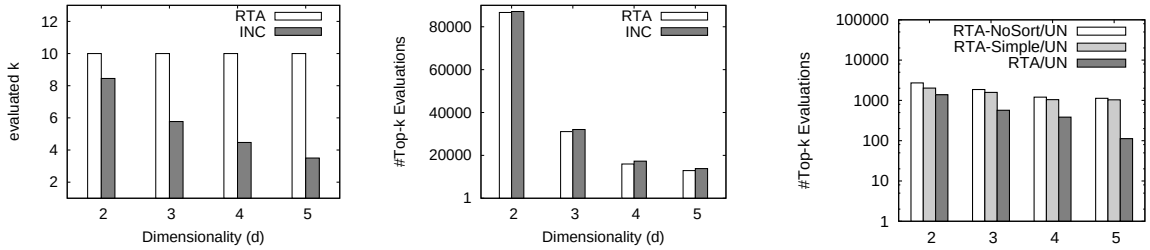
(a) Scalability with $|W|$. (b) Scalability with $|S|$. (c) Scalability with top-k.

Fig. 13. Scalability study of RTOP-Grid for the space-bounded strategy.



(a) Scalability with $|S|$. (b) Scalability with $|W|$. (c) Scalability with top-k.

Fig. 14. Performance of monochromatic algorithm vs. RTA.



(a) Evaluated value of k. (b) Number of top-k evaluations.

Fig. 15. Incremental threshold refinement vs. RTA.

Fig. 16. Effect of sorting W.

7.4 RTA with Incremental Threshold Refinement

In the next experiment (Fig. 15), we study the variant of RTA algorithm that incrementally refines the threshold during each top-k evaluation, as discussed in Section 5.4. We denote this variant of RTA as INC. We use the default setup of uniform data distribution for S and W, $|S|$=10K, $|W|$=10K, top-k=10 and we vary the dimensionality from 2 to 5. In Fig. 15(a), the average value of k used for the top-k queries is shown. RTA always issues top-k queries with k=10. In contrast, INC results in retrieving in each top-k evaluation significantly fewer objects than k. On the other hand, this leads to an increased number of top-k query evaluations performed by INC (Fig. 15(b)). By retrieving fewer than k data objects, the buffer is partially updated, leading to an inaccurate threshold, which in turn triggers more top-k evaluations. Concluding, INC requires a higher number of top-k query evaluations, however retrieving fewer than k data objects on average.

7.5 Effect of Sorted Weighting Vectors

In Fig. 16, we examine the improvement of the performance of RTA caused by the sorting of W based on pairwise similarity. We compare RTA against a version that does not employ sorting (RTA-NoSort) and accesses the vectors in a random order. In addition, we test the performance of sorting based on similarity to a predetermined vector (RTA-Simple), namely the diagonal vector of the space. We evaluate all approaches with weighting vectors that follow a uniform data distribution. The results show that depending on the dimensionality RTA requires up to one order of magnitude fewer top-k evaluations than RTA-NoSort. Furthermore, the performance of RTA using pairwise similarity is clearly much better than RTA-Simple, while the latter is only slightly better than RTA-NoSort. This experiment verifies that our proposed sorting based on pairwise similarity is appropriate for the RTA algorithm. Nevertheless, all variants perform much better than naive, which evaluates $|W|$=10K top-k queries.

8 Related Work

As reverse top-k queries are inherently related to top-k query processing, we summarize some representative work here. One family of algorithms are those based on preprocessing techniques. *Onion* [7] precomputes 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* [1] uses materialized views of top-k result sets, according to arbitrary scoring functions. Onion and Prefer are mostly appropriate for static data, due to the high cost of preprocessing. Efficient maintenance of materialized views for top-k queries is discussed in [10]. The *robust index* [2] is a sequential indexing approach that improves the performance of Onion [7] and Prefer [1]. The main idea is that a tuple should be placed at the deepest layer possible, to reduce the probability of accessing it at query processing time, without compromising the correctness of the result. Later, in [11], the *dominant graph* is proposed as a structure that captures dominance relationships between points. 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. [12] introduce TA and NRA algorithms. Variations of them have been proposed leading to various threshold-based algorithms [13], [14], [15], [16].

Reverse nearest neighbor (RNN) queries were originally proposed in [4]. An RNN query finds the set of points that have the query point as their nearest neighbor. Recently, reverse furthest neighbor queries [17] are introduced, that are similar to RNN queries. The reverse skyline query [5], [6] identifies customers that would be interested in a product based on the dominance of the competitors products. DADA [18] aims to help manufactures position their products in the market, based on three types of dominance relationship analysis queries. Creating competitive products has been recently studied in [19]. 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. [20] study a different problem, again from the perspective of manufacturers. The authors propose an algorithm that selects the subset of attributes that increases the visibility of a new product. Finding the most influential products with reverse top-k queries has been studied in [21].

9 Conclusions

In this paper, we introduce the reverse top-k query which retrieves all weighting vectors for which the query point belongs to the top-k result set. The proposed query type is important for market analysis and for estimating the impact of a product based on the user preferences and the competitors products. We study two versions of reverse top-k queries, namely monochromatic and bichromatic. For the monochromatic reverse top-k query only a dataset of products is given, while for the bichromatic reverse top-k a set of weighting vectors is also available. The experimental evaluation demonstrates the efficiency of the proposed algorithms.

References

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

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

[3] A. Vlachou, C. Doulkeridis, Y. Kotidis, and K. Nørvåg, "Reverse top-k queries," in *ICDE*, 2010, pp. 365–376.

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

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

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

[7] 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 *SIGMOD*, 2000, pp. 391–402.

[8] D. J. Rosenkrantz, R. E. Stearns, and P. M. Lewis II, "An analysis of several heuristics for the traveling salesman problem," *SIAM Journal on Computing*, vol. 6, no. 3, pp. 563–581, 1977.

[9] S. Börzsönyi, D. Kossmann, and K. Stocker, "The skyline operator." in *ICDE*, 2001, pp. 421–430.

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

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

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

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

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

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

[16] 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.

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

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

[19] Q. Wan, R. C.-W. Wong, I. F. Ilyas, M. T. Özsu, and Y. Peng, "Creating competitive products," *PVLDB*, vol. 2, no. 1, pp. 898–909, 2009.

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

[21] A. Vlachou, C. Doulkeridis, K. Nørvåg, and Y. Kotidis, "Identifying the Most Influential Data Objects with Reverse top-k queries," in *PVLDB*, vol. 3, no. 1, pp. 364–372, 2010.