Branch-and-Bound Algorithm for Reverse Top-k Queries

Akrivi Vlachou[1,2], Christos Doulkeridis[1,3], Kjetil Nørvåg[1] and Yannis Kotidis[4]
[1]Norwegian University of Science and Technology (NTNU), Trondheim, Norway
[2]Institute for the Management of Information Systems, R.C. “Athena”, Athens, Greece
[3]University of Piraeus, Piraeus, Greece
[4]Athens University of Economics and Business (AUEB), Athens, Greece
{vlachou,cdoulk,noervaag}@idi.ntnu.no, kotidis@aueb.gr

ABSTRACT

Top-k queries return to the user only the k best objects based on the individual user preferences and comprise an essential tool for rank-aware query processing. Assuming a stored data set of user preferences, reverse top-k queries have been introduced for retrieving the users that deem a given database object as one of their top-k results. Reverse top-k queries have already attracted significant interest in research, due to numerous real-life applications such as market analysis and product placement. Currently, the most efficient algorithm for computing the reverse top-k set is RTA. RTA has two main drawbacks when processing a reverse top-k query: (i) it needs to access all stored user preferences, and (ii) it cannot avoid executing a top-k query for each user preference that belongs to the result set. To address these limitations, in this paper, we identify useful properties for processing reverse top-k queries without accessing each user's individual preferences nor executing the top-k query. We propose an intuitive branch-and-bound algorithm for processing reverse top-k queries efficiently and discuss novel optimizations to boost its performance. Our experimental evaluation demonstrates the efficiency of the proposed algorithm that outperforms RTA by a large margin.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*Query processing*

General Terms

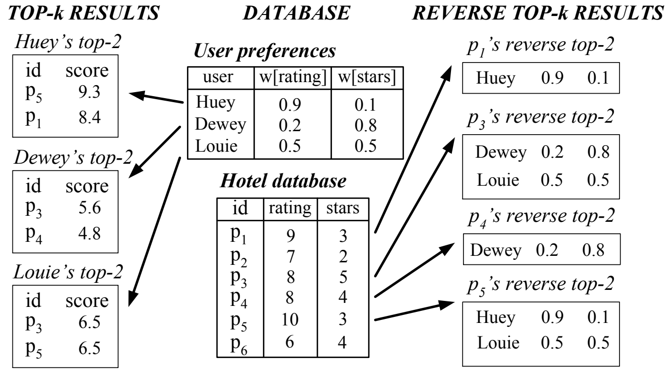
Algorithms, Experimentation, Performance

Keywords

Reverse top-k query, branch-and-bound algorithm

1. INTRODUCTION

Given a database of objects described by a set of numerical scoring attributes and a user with a preference function defined over these attributes, a top-k query retrieves the k objects with best score for the particular preference function. In the model that is widely used in related work [4, 10] and in practice, the 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. Assuming a stored data set of user preferences, reverse top-k queries have been proposed [19, 20] to retrieve the user preferences that make a given object belong to the respective top-k result set. From the perspective of a manufacturer, it is important to identify the customers who are potentially interested in her products and to estimate the visibility of a product based on the different user preferences for which it appears in the top-ranked positions. Hence, reverse top-k queries comprise an essential tool for business analysis, allowing manufacturers to assess the impact of their products in the market based on the competition.



Huey's top-2

id	score
p_5	9.3
p_1	8.4

Dewey's top-2

id	score
p_3	5.6
p_4	4.8

Louie's top-2

id	score
p_3	6.5
p_5	6.5

User preferences

user	w[rating]	w[stars]
Huey	0.9	0.1
Dewey	0.2	0.8
Louie	0.5	0.5

Hotel database

id	rating	stars
p_1	9	3
p_2	7	2
p_3	8	5
p_4	8	4
p_5	10	3
p_6	6	4

p_1's reverse top-2

Huey	0.9	0.1

p_3's reverse top-2

Dewey	0.2	0.8
Louie	0.5	0.5

p_4's reverse top-2

Dewey	0.2	0.8

p_5's reverse top-2

Huey	0.9	0.1
Louie	0.5	0.5

Figure 1: Example of reverse top-k queries.

More formally, a reverse top-k query returns for a point q and a positive integer k, the set of linear preference functions (in terms of weighting vectors) for which q is contained in their top-k result. Consider for example a database containing information about different hotels as well as user preferences, as depicted in Figure 1. For each of the six hotels, the rating and the number of stars are recorded, and maximum values on each attribute are preferable[1]. The database also stores the preferences of three users (Huey, Dewey and Louie) in terms of weights on each attribute. Different users may have different preferences about a potential hotel. For instance, Huey prefers hotels with high rating values, whereas Dewey is interested in hotels with many stars. Louie is indifferent or values equally rating and stars. On the left part of the figure, the top-2 hotels are depicted for each user along with their scores. On the right part, the reverse top-2 results are shown for the hotels. Notice

SIGMOD'13, June 22–27, 2013, New York, New York, USA.
Copyright 2013 ACM 978-1-4503-2037-5/13/06 ...$15.00.

[1] In the remaining of this paper, minimum values will be preferable, without loss of generality.

that p_2 and p_6 have empty reverse top-2 result sets, i.e., they do not belong to the top-2 list of any user.

Currently, the most efficient algorithm for computing the reverse top-k set is the RTA algorithm [19]. RTA has two main drawbacks when processing a reverse top-k query: (i) it needs to access all stored user preferences, and (ii) it cannot avoid executing a top-k query for each user preference (determined by the corresponding user weights) that belongs to the result set. As a result, the performance of RTA is sensitive to the cardinality of the reverse top-k result; for queries with result sets of high cardinality RTA often becomes inefficient. Since we expect that reverse top-k queries will be posed for query points that are highly ranked, and therefore have a result set of high cardinality, this drawback severely limits the practicality of RTA.

To alleviate the shortcomings of RTA, in this paper, we study the conditions in which a set of weighting vectors (representing linear preference functions) can be immediately added to the result set. Therefore, we focus on whether a data point may be ranked higher than the query point for a set of weighting vectors. In addition, we address the question whether a set of weighting vectors can be excluded from the reverse top-k result. Based on these properties, we develop an efficient branch-and-bound algorithm assuming that both data sets are indexed by multidimensional access methods.

The contributions of this paper are summarized here:

- We introduce useful properties for processing reverse top-k queries without accessing each user's individual preferences nor executing the respective top-k query.
- We present a novel algorithm that processes sets of weighting vectors, without having to examine each vector individually, and use this algorithm as basic building block for our reverse top-k algorithms.
- We propose a framework for reverse top-k query processing that employs the branch-and-bound methodology and exploits the introduced properties.
- We present two optimizations of the basic branch-and-bound algorithm (BBR) that use result sharing ($BBR*$) and an aggregate R-tree ($BBRA$) to boost its performance.
- We conduct a thorough experimental evaluation that demonstrates the efficiency of our proposed algorithms.

The rest of this paper is structured as follows: Section 2 reviews related work. Section 3 presents some preliminaries, while Section 4 introduces the theoretical properties. Then, in Section 5, we present how sets of weighting vectors can be processed, without having to examine each one individually. Section 6 describes the branch-and-bound algorithm and its optimizations. In Section 7, we present the results of the experimental evaluation, and we conclude in Section 8.

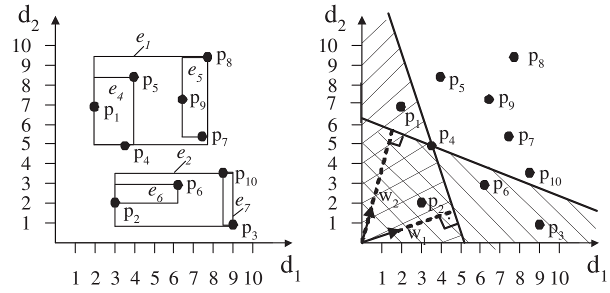
2. RELATED WORK

Recently, the support of efficient top-k query processing has attracted much attention in the database research community. As reverse top-k queries are inherently related to top-k query processing, we summarize some representative work here. Onion [4] 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 [10] 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 query scoring function, and examines a subset of the data elements in this view. Efficient maintenance of materialized views for top-k queries is discussed in [26]. The robust index [24] is a sequential indexing approach that improves the performance of Onion [4] and Prefer [10]. 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. Fagin et al. [7] introduce TA and NRA algorithms for computing the top-k queries over multiple sources, where each source provides a ranking of a subset of attributes only. 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, 5, 9, 14]. Branch-and-bound processing of top-k queries has been studied in [17].

Reverse top-k queries [19, 20] have been proposed for assessing the impact of a potential product in the market, based on the number of users that deem this product as one of their top-k products according to their preferences. Recently, various applications of reverse top-k queries have appeared, including identifying the most influential products [22], and monitoring the popularity of locations based on user mobility [21].

Evaluation of multiple top-k queries has been studied in [8]. Given a data set of points and a set of ranking functions, the authors propose some methods to compute the top-k for all functions (all top-k query). The proposed methods exploit the fact that similar queries share common results to avoid evaluating the top-k queries one-by-one. The first algorithm (BINL) assumes that the data points are indexed by a multidimensional index and the functions are partitioned into groups based on their similarity. BINL processes each group of functions separately and uses bounds in order to avoid computing the exact score of the MBRs. Although BINL reduces the score computations that are required to retrieve the top-k sets of all functions, BINL does not support early termination, which means that the entire index of the data points has to be traversed once for each group of functions. The second algorithm (BLPTA) relies on materialized views. BLPTA answers multiple top-k queries by traversing each view once and can terminate early. In addition, the authors show that the proposed methods can be used also for processing reverse top-k queries. Obviously, if the result set for all top-k queries is known, it is trivial to find the reverse top-k set of a query point. Nevertheless, as shown in the experimental evaluation of [8], computing the all top-k query instead of a reverse top-k query is more expensive and it is useful only if the result of the all top-k query is maintained for answering multiple reverse top-k queries. A recent approach for evaluating multiple top-k queries has appeared in [27]. The proposed framework can be employed to process reverse top-k queries efficiently, however it requires to pre-process all top-k queries in W, and then build an index over the k-th ranked objects of each query. In contrast, our approach does not require such heavy pre-processing, and more importantly it avoids processing all top-k queries.

Different reverse query types, which take as input a data point and aim to find the queries that have this data point in their result set have been studied. Reverse nearest neighbor (RNN) queries were originally proposed in [11]. An RNN query finds the set of points that have the query point as their nearest neighbor. Recently, reverse furthest neighbor queries [25] are introduced, that are similar to RNN queries. The reverse skyline query [6, 13] identifies customers that would be interested in a product based on the dominance of the competitors products. In [3], the authors generalize the concept of reverse queries and propose the inverse queries that take as an input more than one data point. In particular, inverse



(a) Data set S (b) Reverse top-k query

Figure 2: Data set S and reverse top-k query.

range queries, inverse k-nearest neighbor queries, and inverse skyline queries are studied.

Several papers have proposed methods that aim to quantify the impact of products in the market. DADA [12] 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 [23]. 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. [15] 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. Customer identification and product positioning has been recently studied in [2], where the attractiveness of a product is defined based on the concept of reverse skyline query.

3. PRELIMINARIES

Let D denote a data space defined by a set of n dimensions $\{d_1, \ldots, d_n\}$. Let S denote a set of database objects on D with cardinality $|S|$. Each dimension represents a numerical scoring attribute. A database object can be represented as a point $p \in S$, such that $p = \{p[1], \ldots, p[n]\}$, where $p[i]$ is a value on dimension d_i. Thus, the values $p[i]$ are numerical non-negative scores that evaluate the corresponding attributes of database objects. Without loss of generality, in this paper, we further assume that smaller score values are preferable.

In this paper, we assume that the data set S is indexed by a multidimensional index, such as an R-tree. The R-tree groups together nearby points in the data space and represents them by minimum bounding rectangles (MBRs) in the next higher level of the tree. An MBR is the smallest rectangle completely enclosing a set of points and each MBR contains at least one data point. Each MBR m is described by the coordinates of two of its opposite corners, namely the lower-left corner ($m.l$) and the upper-right corner ($m.u$). An intermediate entry e_i corresponds to the minimum bounding rectangle (MBR) $e_i.m$ that encloses the entries of the lower level, while a leaf entry corresponds to a data point. Figure 2(a) depicts an example of a data set S indexed by an R-tree.

3.1 Top-k Queries

Top-k queries are defined based on a scoring function f that aggregates the individual scores of an object into an overall score. 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: $f_w(p) = \sum_{i=1}^{n} w[i] \cdot p[i]$, where $w[i] \geq 0$ $(1 \leq i \leq n)$ and $\sum_{i=1}^{n} w[i] = 1$. Since the weights represent the *relative* importance between different dimensions the assumption $\sum_{i=1}^{n} w[i] = 1$ does not influence the definition of top-k queries.

Symbols	Description
D	Data space
n	Data dimensionality
S	Data set of data points
W	Data set of weighting vectors
V	Subset of W
$p[i]$	Value of point p on dimension i
q	n-dimensional query point
w	Weighting vector
m	Minimum bounding rectangle (MBR)
$\ell_V(m)$	Lower bound of score of m based on V
$u_V(m)$	Upper bound of score of m based on V
$q \prec_V m$	q precedes m based on V
$RTOP_k(q)$	Reverse top-k result set

Table 1: Overview of symbols.

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 $\forall p_i, p_j : p_i \in TOP_k(w)$, $p_j \in S - TOP_k(w)$ it holds that $f_w(p_i) \leq f_w(p_j)$.*

A delicate situation arises when two (or more) objects share the same score for the k-th position. In this case, for simplicity reasons, we assume that it suffices to report any one of them as k-th result.

3.2 Reverse Top-k Queries

Given a query object q, the reverse top-k query identifies all weighting vectors for which q belongs to the top-k result set. The formal definition of the reverse top-k query follows. Notice that this definition corresponds to the bichromatic version of the reverse top-k query (cf. [19]), which assumes that a set of user preferences W exists.

DEFINITION 2. (Reverse top-k query [19])*: Given a point q and a positive number k, as well as two data sets S and W of data points and weighting vectors respectively, a weighting vector $w_i \in W$ belongs to the reverse top-k ($RTOP_k(q)$) result set of q, if and only if $\exists p \in TOP_k(w_i)$ such that $f_{w_i}(q) \leq f_{w_i}(p)$.*

In Figure 2(b), a data set S is depicted together with two different weighting vectors w_1 and w_2. Geometrically, in the Euclidean space a linear top-k query can be represented by a vector w. In this example, the dimension d_1 is preferable for w_1, whereas d_2 is more preferable for w_2. Assuming that p_4 is the query point, the shaded areas define its rank, which is equal to the number of the points enclosed in the corresponding shaded area of a weighting vector w_i. We notice that w_1 belongs to the reverse top-3 query result set, since only 2 objects are contained in the shaded area of w_1. However, w_2 belongs to the reverse top-4 query result set (but not to the reverse top-3), since there exist 3 objects in the shaded area of w_2.

4. THEORETICAL PROPERTIES

In this section we analyze the effect of a set $V \in W$ of weighting vectors on (a) the score of a single data point $p \in S$, and (b) on the score of a set of data points $\{p_i\} \in S$ (represented by an MBR m, i.e., each p_i is enclosed in m). In particular, we derive lower and upper bounds on the scores of p and m. The derived bounds establish score precedence relationships between a query point q and other data point(s). Eventually, this allows us to determine whether the rank of q based on V is affected by a set of data points. Table 1 summarizes the most important symbols used in this paper.

4.1 Score Bounds on Points and MBRs

First, we provide a formal definition of the *score* of a data point based on a weighting vector $w \in W$.

DEFINITION 3. (Score of point p)*: Given a data point $p \in S$ and a weighting vector $w \in W$, the score of p is: $f_w(p) = \sum_{i=1}^{n} w[i] \cdot p[i]$.*

Given a set of weighting vectors $V \subseteq W$, we define the minimum and maximum score of a data point p based on V. We denote the score-lower-bound and score-upper-bound of p with $\ell_V(p)$ and $u_V(p)$ respectively.

DEFINITION 4. (Score-lower-bound of point p)*: Given a set of weighting vectors V and a data point p, the score-lower-bound of p is: $\ell_V(p) = \sum_{i=1}^{n} min_{\forall w \in V}(w[i]) \cdot p[i]$.*

DEFINITION 5. (Score-upper-bound of point p)*: Given a set of weighting vectors V and a data point p, the score-upper-bound of p is: $u_V(p) = \sum_{i=1}^{n} max_{\forall w \in V}(w[i]) \cdot p[i]$.*

Capitalizing on these scores of a point p with respect to a set of weighting vectors V, we are able to derive a *lower bound* and an *upper bound* of the score of p in a straightforward manner.

LEMMA 1. (Score bounds of p)*: Given a set of weighting vectors V and a data point p, the score $f_w(p)$ of p is lower-bounded by $\ell_V(p)$ and upper-bounded by $u_V(p)$, i.e., it holds that $\forall w \in V$: $\ell_V(p) \leq f_w(p) \leq u_V(p)$.*

PROOF. It holds that: $f_w(p) = w[1] \cdot p[1] + \cdots + w[n] \cdot p[n] \leq max_{\forall w \in V}(w[1]) \cdot p[1] + \cdots + max_{\forall w \in V}(w[n]) \cdot p[n] = u_V(p)$. Similarly, it holds that: $f_w(p) = w[1] \cdot p[1] + \cdots + w[n] \cdot p[n] \geq min_{\forall w \in V}(w[1]) \cdot p[1] + \cdots + min_{\forall w \in V}(w[n]) \cdot p[n] = \ell_V(p)$. □

In the following, we generalize the score bounds for the case of a set of data points that are represented by a minimum bounding rectangle (MBR) m and we denote as $m.l$ the lower-left corner and as $m.u$ the upper-right corner. Given a set of weighting vectors $V \subseteq W$ and an MBR m of data points, we can define a lower and upper bound of the scores of all data points that are enclosed in m based on any $w \in V$. The bounds are $\ell_V(m) = \ell_V(m.l)$ and $u_V(m) = u_V(m.u)$ respectively.

LEMMA 2. (Score bounds of MBR m)*: Given a set of weighting vectors $w \in V \subseteq W$ and an MBR $m \in S$, the score $f_w(p)$ of every point $p \in m$ is lower-bounded by $\ell_V(m) = \ell_V(m.l)$ and upper-bounded by $u_V(m) = u_V(m.u)$, i.e., it holds that $\forall p \in m, \forall w \in V$: $\ell_V(m) \leq f_w(p) \leq u_V(m)$.*

PROOF. It holds that: $f_w(p) = w[1] \cdot p[1] + \cdots + w[n] \cdot p[n] \leq max_{\forall w \in V}(w[1]) \cdot p[1] + \cdots + max_{\forall w \in V}(w[n]) \cdot p[n] \leq max_{\forall w \in V}(w[1]) \cdot m.u[1] + \cdots + max_{\forall w \in V}(w[n]) \cdot m.u[n] = u_V(m)$. Similarly, it holds that: $f_w(p) = w[1] \cdot p[1] + \cdots + w[n] \cdot p[n] \geq min_{\forall w \in V}(w[1]) \cdot p[1] + \cdots + min_{\forall w \in V}(w[n]) \cdot p[n] \geq min_{\forall w \in V}(w[1]) \cdot m.l[1] + \cdots + min_{\forall w \in V}(w[n]) \cdot m.l[n] = \ell_V(m)$. □

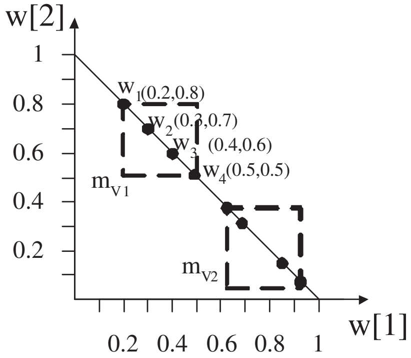


Figure 3: Example of data set W.

In the case in which an MBR m_V that encloses the set of weighting vectors V is given, the definition of the bounds $\ell_V(p)$ and $u_V(p)$ of the scores for a data point p can be derived by means of the lower-left corner and the upper-right corner of the MBR m_V respectively.

EXAMPLE 1. *In Figure 3, a set $V = \{w_1, w_2, w_3, w_4\}$ of four 2-dimensional weighting vectors is depicted, which are enclosed by an MBR defined by the lower-left corner $[0.2, 0.5]$ and the upper-right corner $[0.5, 0.8]$. Then, the values $\ell_V(p) = 0.2 \cdot p[1] + 0.5 \cdot p[2]$ and $u_V(p) = 0.5 \cdot p[1] + 0.8 \cdot p[2]$ define a lower and an upper bound respectively of the score $f_w(p)$ of any data point p based on V. For example, consider a data point p with $p[1] = 5$ and $p[2] = 3$, then $f_{w_1}(p) = 3.4$, $f_{w_2}(p) = 3.6$, $f_{w_3}(p) = 3.8$ and $f_{w_4}(p) = 4$, while $\ell_V(p) = 2.5$ and $u_V(p) = 4.9$.*

4.2 Score Precedence

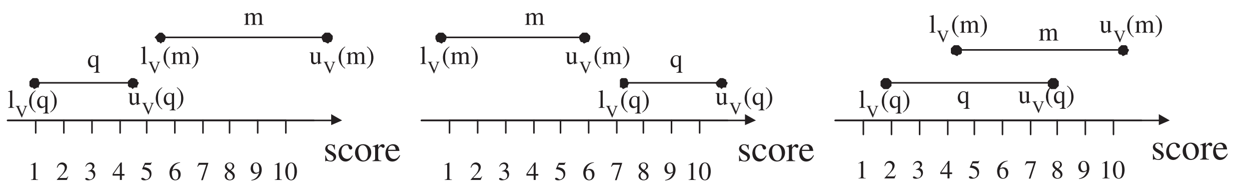
Consider a reverse top-k query, defined by a query point q and a value k, and a multidimensional index (R-tree) over data set S. The goal is to find under which conditions a set of weighting vectors $V \subseteq W$ belongs to the reverse top-k result of q. In order to solve this problem, we first determine whether q has a better rank than a set of data points that are enclosed in an MBR m.

Given a set $V \subseteq W$ of weighting vectors, we can exploit the score bounds of data points and MBRs in order to map their potential scores to intervals. For instance, $\forall w \in V$ the score $f_w(q)$ of the query point belongs to the interval $[\ell_V(q), u_V(q)]$. Similarly, the score $f_w(p)$ of every point $p \in m$ belongs to the interval $[\ell_V(m), u_V(m)]$. We define a partial order between the query point and the MBRs of S based on the score bounds, which is used to compare the potential score of q to the score of the MBRs.

DEFINITION 6. (Score precedence)*: Given a set $V \subseteq W$ of weighting vectors, a query point q, and an MBR m of data points, if $u_V(q) < \ell_V(m)$, then we say that q precedes m and we denote $q \prec_V m$. Similarly, if $u_V(m) < \ell_V(q)$, we say that m precedes q and we denote $m \prec_V q$. Otherwise, we say that q and m are incomparable and we denote $m \asymp_V q$.*

By studying the possible precedence relationships between the score interval of q and of an MBR m, we distinguish the following three cases (as depicted in Figure 4):

- **Case 1:** If $u_V(q) < \ell_V(m)$, i.e., q precedes m ($q \prec_V m$), then $\forall w \in V$ the score $f_w(q)$ of q is better than the score $f_w(p)$ of every point $p \in m$. Consequently, no point $p \in m$ can affect the rank of q for any weighting vector $w \in V$.



(a) Case 1: q precedes m ($q \prec_V m$)

(b) Case 2: m precedes q ($m \prec_V q$)

(c) Case 3: q, m incomparable ($m \asymp_V q$)

Figure 4: Intervals of scores and precedence relationships between query point q and MBR m.

- **Case 2:** If $u_V(m) < \ell_V(q)$, i.e., m precedes q ($m \prec_V q$), then every data point $p \in m$ has a better score than q for any weighting vector $w \in V$. This means that all points $p \in m$ have a better rank than q for any weighting vector $w \in V$.

- **Case 3:** If none of the above cases holds, then q and m are incomparable ($m \asymp_V q$) and there exists an overlap between the intervals. In this case, it cannot be determined whether q has a better or worse score than a data point $p \in m$ for a weighting vector $w \in V$. This means that q can have a better rank than some points of m for a weighting vector $w \in V$, while it can have a worse rank for other points of m or another $w' \in V$.

5. INTOPK ALGORITHM

To address limitations of existing approaches, the first step is to provide a method for determining if a given set of weighting vectors $V \subseteq W$ belongs to the reverse top-k results of q or not. To this end, we present the INTOPk algorithm that efficiently answers this question. Intuitively, INTOPk attempts to handle a set of weighting vectors without accessing each individual vector nor processing the top-k query, thus providing the missing functionality of RTA. This constitutes a key technical contribution of this paper. In the next section, we will describe how INTOPk can be exploited by a branch-and-bound algorithm that can directly compete with RTA.

In the following, a set $V \subseteq W$ of weighting vectors is represented by an MBR denoted as m_V. Given a data set S, the INTOPk algorithm takes as an input the MBR m_V, a query point q and a value k, and returns whether q belongs to the top-k result set for all or none of the weighting vectors enclosed in m_V. If none of these cases holds, the INTOPk algorithm returns that it cannot give a definite answer.

5.1 Pruning Property and Result Inclusion

By exploiting score precedence between the query point q and an MBR m_V of weighting vectors V, we can derive a useful pruning property for sets of weighting vectors $V \subseteq W$. The pruning property enables discarding MBRs of weighting vectors at once, without even accessing the individual vectors.

THEOREM 1. (Pruning property)*: Given a set of weighting vectors $V \subseteq W$ represented by an MBR m_V, and a reverse top-k query $RTOP_k(q)$, if k data items (MBRs or data points) precede q based on V, then m_V can be safely pruned, i.e., no weighting vector $w \in V$ belongs to the reverse top-k result of q.*

PROOF. By contradiction. Let us assume that $w \in V$ belongs to the reverse top-k result of q. Since k data items precede q based on V and any MBR encloses at least one data point, it holds that $\exists p_i$, $1 \leq i \leq k$ such that $u_V(p_i) < \ell_V(q)$, which leads to $f_w(p_i) \leq u_V(p_i) < \ell_V(q) \leq f_w(q)$. This means that there exist at least k data points with better score than q for any $w \in W$. This leads to a contradiction, since by definition if w belongs to the reverse top-k result of q, then at most $k-1$ data points p_i have a better score than q based on w. □

Additional pruning can be achieved in the case that V consists of a single vector w. Again, w can be discarded without evaluating the top-k query, based on the following lemma.

LEMMA 3. *Given a single weighting vector $w \in W$ and a reverse top-k query $RTOP_k(q)$, if for at least k data points p_i it holds that $u_{\{w\}}(q) > \ell_{\{w\}}(p_i)$, then the weighting vector $w \in W$ can be safely excluded from the reverse top-k result of q.*

PROOF. Let p_i, $1 \leq i \leq k'$ be the k' ($\geq k$) data points such that $u_{\{w\}}(q) > \ell_{\{w\}}(p_i)$. Moreover, for any data point by definition it holds that $\ell_{\{w\}}(p) = f_w(p) = u_{\{w\}}(p)$. Thus, $\forall p_i$ it holds that: $f_w(p_i) = \ell_{\{w\}}(p_i) < u_{\{w\}}(q) = f_w(q)$. Since for at least k data points it holds that $f_w(p) < f_w(q)$ (k data points have a better score than q for w), it means that w is not in the reverse top-k result of q. □

Notice that Lemma 3 does not hold for MBRs of weighting vectors, but only for a single weighting vector.

Furthermore, we derive a rule for immediate inclusion of an MBR of weighting vectors to the reverse top-k query result, without examining each weighting vector individually.

THEOREM 2. (Result inclusion)*: Given a set of weighting vectors $V \subseteq W$ represented by an MBR m_V, and a reverse top-k query $RTOP_k(q)$, if fewer than k data points p_i exist such that $u_V(q) > \ell_V(p_i)$, then all weighting vectors $w \in V$ can be safely added to the reverse top-k result of q.*

PROOF. Let p_i, $1 \leq i \leq k' < k$ be the k' data points such that $u_V(q) > \ell_V(p_i)$. Then, for all remaining points $p \in S - \{p_i\}$ it holds that $u_V(q) \leq \ell_V(p)$. We will prove the theorem by contradiction. Let us assume that $\exists w \in V$ that does not belong to the reverse top-k result of q. This means that there exist k data points p'_i such that $f_w(p'_i) < f_w(q)$ from which we derive that $\ell_V(p'_i) \leq f_w(p'_i) < f_w(q) \leq u_V(q)$, i.e., $\ell_V(p'_i) < u_V(q)$. Since $\forall p \in S - \{p_i\}$ it holds that $u_V(q) \leq \ell_V(p)$ and the size of set $\{p_i\}$ is k' ($< k$), this leads to a contradiction. □

The aforementioned theorems 1 and 2 and lemma 3 guide the design of an efficient algorithm for determining whether all weighting vectors in V or none of them belong to the reverse top-k results set of q.

5.2 Algorithmic Description

Algorithm 1 describes the pseudocode for the INTOPk algorithm. The algorithm contains an initialization phase, which is followed by a traversal of the R-tree of data set S. The algorithm uses two counters to determine whether m_V belongs to the result set or

Algorithm 1 INTOPk()

```
1:  Input: MBR $m_V$ of weighting vectors, query $RTOP_k(q)$
2:  Output: -1: discard $m_V$, 0: inconclusive, 1: add $m_V$
3:  $precincPoints \leftarrow 0, precEntries \leftarrow 0$
4:  $e \leftarrow$ RtreeS.getRoot()
5:  if ($u_V(q) > \ell_V(e.m)$) then
6:      heapS.enqueue(e)
7:      if ($e.m \prec_V q$) then
8:          $precEntries$++
9:  while (!heapS.isEmpty()) do
10:     $e \leftarrow$ heapS.dequeue()
11:     if (($precincPoints \geq k$) and ($\ell_V(e.m) \geq \ell_V(q)$)) then
12:         if ($m_V$ is a single $w$) then
13:             return -1
14:         else
15:             return 0
16:     if ($e.m \prec_V q$) then
17:         $precEntries$– –
18:     $\mathcal{C} \leftarrow expand(e)$
19:     for all ($e_i \in \mathcal{C}$) do
20:         if ($u_V(q) > \ell_V(e_i.m)$) then
21:             if ($e_i.m \prec_V q$) then
22:                 $precEntries$++
23:                 if ($precEntries \geq k$) then
24:                     return -1
25:             if ($e_i$ is a data point) then
26:                 $precincPoints$++
27:             else
28:                 heapS.enqueue($e_i$)
29: if ($precincPoints \geq k$) then
30:     return 0
31: else
32:     return 1
```

not. The first counter denoted as $precEntries$ counts the entries e such that $e.m$ precede q based on V. If the number of these entries is greater than or equal to k, then we can exclude m_V from the result set based on Theorem 1. The second counter $precincPoints$ counts the data points that are either incomparable or precede q. Based on Theorem 2, if fewer than k such entries exist, m_V can be added to the result set[2].

In the initialization phase (lines 3–8), the root e of the R-tree is accessed. If $u_V(q) > \ell_V(e.m)$, then some of the data points enclosed in $e.m$ may have a better rank than q for some weighting vector enclosed in m_V. Therefore, the root entry is inserted into a heap (*heapS*) that contains entries of S that need to be further examined. This heap is sorted in ascending order of $\ell_V(m)$, because the entries with small (better) lower bound values have a higher probability to discard m_V. Then, the algorithm tests if $e.m$ precedes q. If that is the case, then all points enclosed in $e.m$ have a better rank than q for all weighting vectors in m_V. Since we assume that at least one data point is enclosed in each MBR $e.m$, $precEntries$ increases by one. Note that if q precedes the root entry $e.m$, then q has a better rank than any point of S, and the algorithm returns 1 (line 32) indicating that all weighting vectors of m_V can be added to the result set.

During the traversal of the R-tree (lines 9–28), in each iteration, the entry e of the heap with the best $\ell_V(m)$ value is examined. The INTOPk algorithm excludes m_V from the result set whenever it can safely decide that none of the weighting vectors in m_V can be included to the result set. Thus, if k data points have been retrieved that are either incomparable or precede q (line 11), it checks whether an MBR $e.m$ that has not been examined yet can exclude m_V from the result set. Since the MBRs are examined sorted based on $\ell_V(m)$, if the value $\ell_V(m)$ of the next MBR is larger or equal to $\ell_V(q)$, it means that there exists no other MBR that precedes q and m_V cannot be excluded from the result set. In addition, since k data points exist that are incomparable or precede q, m_V cannot be added to the result set. Therefore, INTOPk returns that it cannot give a definite answer, i.e., inconclusive. On the other hand, if m_V represents a single weighting vector (line 12), then based on Lemma 3 m_V can be discarded.

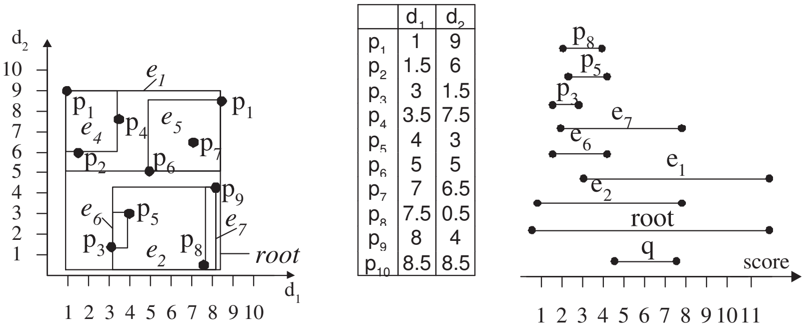
If $precincPoints$ is smaller than k, then the entry e will be expanded. If $e.m$ precedes q based on V, then we reduce $precEntries$ (line 17), since e will be expanded and the enclosed entries (MBRs or data points) will be counted. Since e is a non-leaf entry (only non-leaf entries are added to the heap), we expand the MBR and retrieve the list of children entries $\mathcal{C}$. If q precedes a child entry, then this child entry is ignored, since it cannot influence the ranking of q. Note that also if $u_V(q) = \ell_V(e_i.m)$ the entry can be ignored, because based on the definition only points that have a better score influence the ranking of q. Otherwise, for each $e_i.m$ that precedes q, we increase $precEntries$ by one (line 22). In the case, where $e_i.m$ is a leaf entry, then we test if q precedes $e_i.m$ and if this is not the case we increase $precincPoints$ by one. Non-leaf entries are added to the heap and a new iteration of the algorithm starts. If all entries of S have been either discarded or examined and fewer than k data points that are incomparable or precede q have been found, then all weights of m_V can be added to the result set (line 32).

EXAMPLE 2. *In Figure 5, consider a reverse top-k query with* $k = 2$ *where the query point q has coordinates* $q[1] = 5$ *and* $q[2] = 6.5$. *Furthermore, let* m_V *be defined by the lower-left corner* $m_V.l$=$[0.2, 0.5]$ *and the upper-right corner* $m_V.u$=$[0.5, 0.8]$. *Then, the lower and upper bound of q are* $\ell_V(q)$=*4.25 and* $u_V(q)$=*7.7, respectively. In the initialization phase, the root of the R-tree is accessed. Then, in the first iteration of INTOPk the root entry is expanded and entries* e_1 *and* e_2 *are inserted in* $heapS$. *Since* e_2 *has a smaller* ℓ_V *value, it is accessed first. During the expansion of* e_2, *entries* e_6 *and* e_7 *are inserted in the heap. Next,* e_6 *is accessed (based on the* ℓ_V *value) and the data points* p_3 *and* p_5 *are retrieved. The values of the counters become* $precEntries = 1$ *and* $precincPoints = 2$. *In the next iteration,* e_7 *is accessed and* $precincPoints = 2 \geq k$, *but the INTOPk algorithm continues with the next iteration. This is because* $\ell_V(e_7.m) < \ell_V(q)$ *and entries that may discard* m_V *can still be retrieved. Afterwards,* e_7 *is expanded and the data points* p_8 *and* p_9 *are retrieved. Point* p_8 *causes the increase of* $precEntries = 2$ *and therefore INTOPk terminates by returning -1 indicating that none of the weighting vectors of* m_V *belongs to the reverse top-2 result set of q.*

6. BRANCH-AND-BOUND ALGORITHM

In this section, we present the branch-and-bound algorithm for efficiently retrieving the reverse top-k result set of a query point q. First, we present the basic version of the algorithm in Section 6.1, and then we study two optimizations that boost its performance. In Section 6.2, we employ result sharing of consecutive INTOPk evaluations, in order to reduce the number of invocations that are required. Finally, in Section 6.3, we propose an algorithm that employs an aggregate R-tree, thus resulting in additional performance gains.

[2] For simplicity, we assume that q does not belong to S, but it is straightforward to adapt the algorithm for this case too.



	d1	d2
p1	1	9
p2	1.5	6
p3	3	1.5
p4	3.5	7.5
p5	4	3
p6	5	5
p7	7	6.5
p8	7.5	0.5
p9	8	4
p10	8.5	8.5

Figure 5: Example of INTOPk algorithm.

6.1 Basic Branch-and-Bound Algorithm

Algorithm 2 describes our basic branch-and-bound algorithm (BBR). The algorithm uses an R-tree to index the data set of weighting vectors W. Intuitively, as our algorithm traverses this index, it bounds the search space of results for the reverse top-k query by discarding MBRs of weighting vectors that cannot contribute to the result set. Essentially, each time an entry of the R-tree is processed, our algorithm tests whether the weighting vectors that are enclosed by the MBR of the entry, forming a set V, may belong to the result set or whether it can be discarded. This test is accomplished in practice by using INTOPk algorithm, as described in Section 5. If INTOPk algorithm is inconclusive about V, the current entry of the R-tree needs to be expanded, and smaller subsets of V need to be examined for inclusion or pruning. The goal of BBR is to expand as few entries as possible by either discarding entries of the R-tree or by adding them to the result set immediately.

BBR traverses the R-tree that indexes the data set W and maintains a heap (*heapW*) of R-tree entries. Initially, the root of the R-tree is inserted in *heapW*. The entries in the heap are sorted based on their distance to a given weighting vector, for example their distance to the vector $w[i] = 1/n$, $\forall i$. Notice that the benefits of accessing weighting vectors based on their similarity to the given vector has been investigated and evaluated experimentally in [20]. Then, in each iteration an entry e (MBR) is accessed from the heap and the INTOPk algorithm is executed for its MBR $e.m$. Depending on the result of INTOPk algorithm, e is either added to the result set, or expanded or discarded. In the case that an entry is expanded, all its children entries are inserted in the heap (line 8). Otherwise, if the weighting vectors that correspond to an entry should be added, then all data points (i.e., weighting vectors) stored in the subtree rooted at e are retrieved and added to the result set (line 11). The $expandAll()$ method in the pseudocode retrieves all weighting vectors stored in the subtree rooted at e.

THEOREM 3. (Correctness of BBR)*: The BBR algorithm always produces the correct reverse top-k result set.*

PROOF. *(Sketch)*: In BBR, every entry e of the R-tree indexing W is either added to the results or discarded (we note that expanding an entry also eventually results in adding or discarding the enclosed entries). Hence, it suffices to show that BBR never discards an entry that contains a weighting vector that is a reverse top-k result (false negatives), and never adds an entry that contains a weighting vector that is not a reverse top-k result (false positives).

- False negatives: Algorithm 1 decides to discard an entry e in two cases. In the first case, k discrete entries $e_i.m$ were found that precede q based on V and Theorem 1 verifies that no vector $w \in V$ can belong to the result. In the second case, only single weighting vectors are discarded, for which it holds that there exist at least k data points p_i such that $u_{\{w\}}(q) > \ell_{\{w\}}(p_i)$, and Lemma 3 proves that these weighting vectors can be safely discarded.
- False positives: Algorithm 1 adds an entry e to the result set, if fewer than k data points p_i exist such that $u_V(q) > \ell_V(p_i)$. Theorem 2 proves that in this case, any $w \in V$ belongs to the reverse top-k result.

□

Algorithm 2 BBR

```
1: Input: Point $q$, value $k$
2: Output: Reverse top-$k$ result set $RTOP_k(q)$
3: heapW.enqueue(RtreeW.getRoot())
4: while (!heapW.isEmpty()) do
5:     $e \leftarrow$ heapW.dequeue()
6:     $i \leftarrow INTOPk(e.m, q, k)$
7:     if $i = 0$ then
8:         heapW.enqueue(expand($e$))
9:     else
10:        if $i = 1$ then
11:            $RTOP_k(q) \leftarrow RTOP_k(q) \bigcup$ expandAll($e$)
12: return $RTOP_k(q)$
```

6.2 Branch-and-Bound with Result Sharing

As BBR traverses the R-tree indexing W, for each entry that needs to be processed, a traversal of the R-tree indexing S is initiated. Clearly, the performance of BBR depends on the number of invocations of the INTOPk algorithm, which is the cause of I/Os on the index of data set S. To improve the performance of BBR, it is beneficial to avoid INTOPk invocations (and the respective I/Os) when possible. Therefore, we employ a *result sharing* method that greatly reduces the accesses on this index. The new result sharing approach is termed $BBR*$ and is depicted in Algorithm 3.

$BBR*$ exploits previously computed results to avoid redundant processing. As explained in Section 5, Algorithm 1 achieves to discard an entry m_V due to the retrieval of k data items m_i (data points or MBRs) of the index on S that precede q on m_V. These data items can potentially lead to discarding subsequent weighting vectors, i.e., if m_i precede q based on the next entry m'_V. Therefore $BBR*$ maintains this set of MBRs in a list of bounded size k, denoted as *topk*, and uses this list for avoiding invocations of the INTOPk algorithm.

Algorithm 3 $BBR*$

```
1: Input: Point $q$, value $k$
2: Output: Reverse top-$k$ result set $RTOP_k(q)$
3: $heapW$.enqueue(RtreeW.getRoot())
4: while (!$heapW$.isEmpty()) do
5:    $m_V \leftarrow heapW$.dequeue()
6:    $precEntries \leftarrow 0$
7:    for all ($m_i \in topk$) do
8:       if ($u_V(m_i) \leq \ell_V(q)$) then
9:          $precEntries$++
10:   if ($precEntries < k$) then
11:      $i \leftarrow INTOPk(m_V, q, k)$
12:      if $i = 0$ then
13:         $heapW$.enqueue(expand($m_V$))
14:      else
15:         if $i = 1$ then
16:            $RTOP_k(q) \leftarrow RTOP_k(q) \bigcup$ expandAll($m_V$)
17:         else
18:            update $topk$
19: return $RTOP_k(q)$
```

In the pseudocode of Algorithm 3, each time an entry m_V is accessed (line 5), a test is conducted between q and each of the MBRs maintained in the list $topk$ (lines 6– 9). If k MBRs precede q (line 8), then the INTOPk invocation based on m_V is avoided (line 10). The list $topk$ is updated each time the INTOPk algorithm results in discarding an entry m_V (line 18).

6.3 Branch-and-Bound with Aggregate R-tree

The aggregate R-tree (aR-tree) [16] is a variant of the R-tree that combines indexing with pre-aggregation. In the aggregate R-tree, each data object is associated with a score. In addition, each entry is annotated with a score value that aggregates the scores of all children entries. Even though different aggregation functions can be employed by an aggregate R-tree, we assume that each entry adds the scores of its children entries. Furthermore, the score of each data object is set equal to one. Thus, the score of each entry corresponds exactly to the number of all points contained in its subtree (i.e., all points enclosed by the entry's MBR).

By employing an aggregate R-tree for indexing the dataset S, the performance of the branch-and-bound reverse top-k algorithm can be improved even further. In Algorithm 1, the counter $precEntries$ estimates the number of points that precede q. Increasing the counter by one is a lower bound of the actual number of data points, since each MBR is assumed to enclose at least one data point. In the case of the aggregate R-tree, the exact number of points that are stored in the subtree rooted by an entry e_i is known. The only necessary modification in Algorithm 1 is to replace the increment (decrement) of $precEntries$ by one, with an increment (decrement) by the score of the respective entry (lines 8, 17 and 22). Then, $precEntries$ provides an exact value, rather than an estimate. By using the aggregate score, Algorithm 1 can determine faster whether m_V should be discarded, and thus fewer entries of the R-tree index on S are expanded. Similarly, the performance of $BBR*$ can be improved by the aggregate R-tree, if in Algorithm 3 the counter $precEntries$ is increased (line 9) based on the aggregate score of the respective entry.

7. EXPERIMENTAL EVALUATION

In this section, we present an extensive experimental evaluation of the proposed branch-and-bound algorithm for reverse top-k queries. All variants of the presented algorithms are implemented in Java and the experiments run on a server with 2x4 cores (AMD Opteron), 32GB RAM, and 2TB HDD. The block size is set to 4KB and the buffer of the R-tree has a size of 100 nodes.

Parameter	**Values**
Data dimensionality n	2,3,**4**,5,6,7,8,9
Data set S distribution	**UN**,CO,AC,CL,RE
Data set cardinality $\|S\|$	**100K**,1M,5M
Data set W distribution	**UN**,CL
Weights cardinality $\|W\|$	**100K**,1M
#queries	1000
Queries selected from	**k-skyband**,skyline
top-k	**10**,20,30,40,50
#clusters for W,S: C_W,C_S	5
Variance σ_W^2, σ_S^2	0.05^2

Table 2: Experimental parameters.

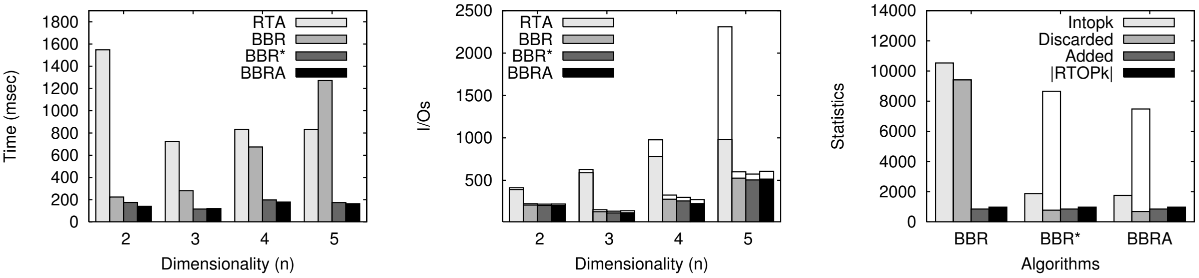
7.1 Experimental Setup

Data sets. In the experimental study, we employ both real (RE) and synthetic data sets, namely uniform (UN), correlated (CO), anti-correlated (AC), and clustered (CL). In the case of synthetic data sets, the generated values of each attribute belong to $[0, 10K]$. For the uniform data set, all attribute values are generated independently using a uniform distribution. The anti-correlated data set 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 data set, 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 data sets. 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. COLOR is a data set from UCI Machine Learning Repository that contains image features extracted from a Corel image collection. It consists of 68040 9-dimensional tuples describing features of images in HSV color space. Both data sets have been used before in research related to rank-aware query processing [18].

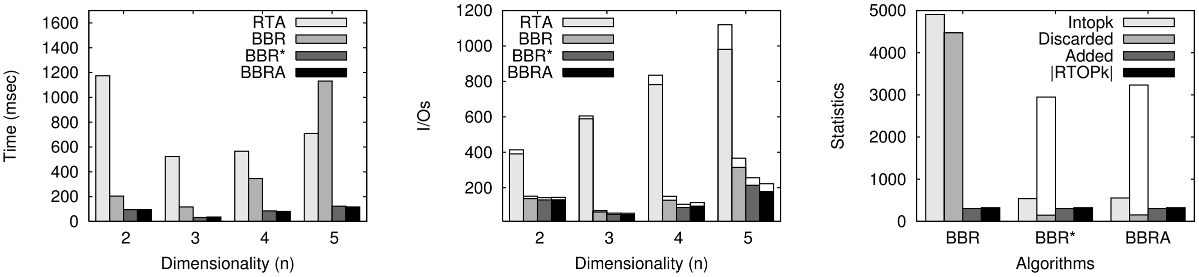
For the data set W of the weighting vectors, two different data distributions are examined, namely uniform (UN) and clustered (CL). For the clustered data set W, first C_W cluster centroids that belong to the (n-1)-dimensional hyperplane defined by $\sum w_i = 1$ are selected randomly. Then, each coordinate is generated on the (n-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.

Description of RTA. For comparative purposes, we evaluate the performance of the proposed branch-and-bound algorithm against the state-of-the-art algorithm for reverse top-k queries, named RTA [19, 20]. RTA works by examining each weighting vector and trying to avoid processing of the respective top-k, when RTA can safely decide that the vector cannot belong to the reverse top-k result. To achieve this, RTA uses a buffer of k entries to keep the results of the previously posed top-k query, and uses these results to prune the next weighting vectors. In RTA, the data set S is indexed by an R-tree and the underlying top-k processing is performed using a state-of-the-art branch-and-bound algorithm for top-k queries. The weighting vectors are examined in an order



(a) Time. (b) I/Os. (c) Statistics.

Figure 6: Comparative performance of all algorithms for UN data set and varying dimensionality (n).



(a) Time. (b) I/Os. (c) Statistics.

Figure 7: Comparative performance of all algorithms for CL data set and varying dimensionality (n).

based on pairwise similarity, to increase the probability of pruning consecutive vectors. Hence, in the case of RTA, W is stored sorted on disk, while S is indexed by an R-tree.

Algorithms. We evaluate three variants of our branch-and-bound algorithm: basic (BBR), with result sharing ($BBR*$), and using aggregate R-tree ($BBRA$). The variants of the branch-and-bound algorithm use R-tree indexes on S and W, while $BBRA$ uses an aggregate R-tree on S.

Metrics. Our main metrics include: a) the query execution time required by each algorithm, and b) the I/Os used. The I/Os induced on S are buffered I/Os, while for the I/Os on W buffering is useless, since for a given query, W is not accessed multiple times. In the charts showing I/Os, each bar corresponds to the total number of I/Os induced by an algorithm, while the white part of the bar shows the I/Os induced on S and the colored part of the bar shows the I/Os induced on W. We also show the results of various statistics measured that clearly explain the behavior of each algorithm. We present average values over 1000 queries in all cases. The query points are randomly selected from a subset of the data points in S. To increase the probability of having non-empty result sets, this subset is either the k-skyband or the skyline set of S.

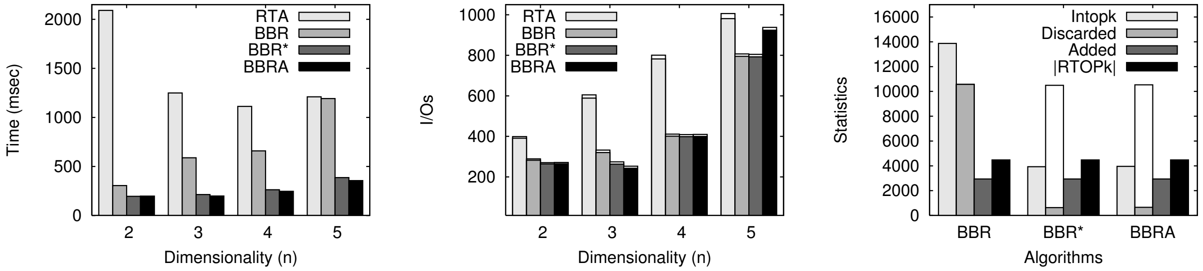
Parameters. We conduct experiments varying the dimensionality n (2-9), the cardinality $|S|$ (100K-5M), the cardinality $|W|$ (100K-1M), the value of k (10-50), and the data distributions for S and W. The default setup is: n=4, $|S|$=100K, $|W|$=100K, k=10, C_W=C_S=5, σ_W^2=σ_S^2=0.05^2, and we use UN distribution for both S and W. The experimental parameters together with default values (in bold) are shown in Table 2.

7.2 Experimental Results

Varying dimensionality n. Figure 6 presents the comparative performance of all algorithms for uniform data distributions of S and W, for varying n, and the default setup: $|S|$=100K, $|W|$=100K, k=10. In terms of execution time (Figure 6(a)), $BBR*$ and $BBRA$ improve the performance of RTA by a factor of 4 to 8, depending on the dimensionality. RTA is inefficient in the lower dimensions and improves its performance as the dimensionality increases. This is mainly because the performance of RTA depends on the cardinality of the reverse top-k result set, which gets smaller as the dimensionality increases (for $n = 2 : |RTOP_k(q)| = 13679.94$ while for $n = 4 : |RTOP_k(q)| = 32.31$). In contrast, the performance of BBR decreases as the dimensionality increases, because the processing of INTOPk becomes more expensive for higher dimensions. On the other hand, $BBR*$ and $BBRA$ scale with dimensionality.

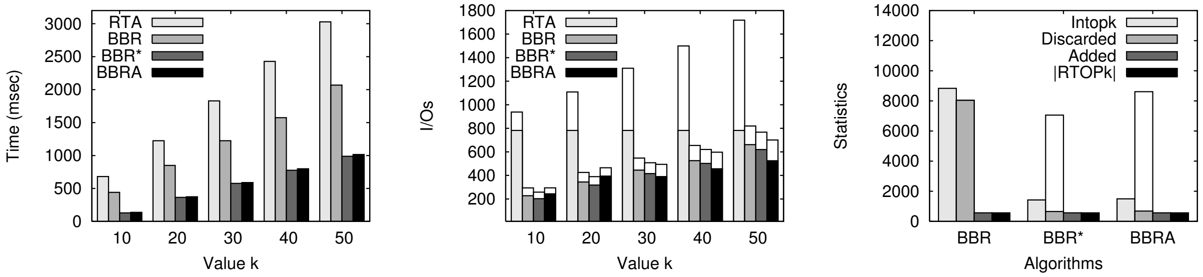
In terms of I/Os (Figure 6(b)), again $BBR*$ and $BBRA$ are up to 4 times better than RTA. We stress that each bar corresponds to the total number of I/Os, while the white part of the bar shows the I/Os induced on S and the colored part of the bar shows the I/Os induced on W. This cost breakdown analysis shows that RTA uses more I/Os both for S and W. The branch-and-bound algorithms are very efficient in terms of I/Os on S due to the buffering employed. However, the important finding is that they all reduce the I/Os on W compared to RTA, which demonstrates the efficiency of the employed score bounds.

Figure 6(c) helps deriving some interesting insights about the branch-and-bound algorithms. The first bar shows the number of INTOPk evaluations for each algorithm. Both $BBR*$ and $BBRA$ consistently require fewer INTOPk evaluations that BBR. The second bar shows the number of discarded MBRs by each algorithm. For $BBR*$ and $BBRA$ the white part of the bar shows the number of MBRs discarded due to result sharing, i.e., without invoking INTOPk. This clearly demonstrates the value of result sharing. The third bar depicts the number of MBRs that were immediately added to the results by INTOPk. The significant finding is that all algorithms manage to add groups of weighting vectors to the result set,



(a) Time. (b) I/Os. (c) Statistics.

Figure 8: Comparative performance of all algorithms for CO data set and varying dimensionality (n).



(a) Time. (b) I/Os. (c) Statistics.

Figure 9: Comparative performance of all algorithms for UN data set and varying value of k.

without examining each weighting vector individually. This becomes clear if we compare the cardinality of the result set (fourth bar), with the added elements (third bar). Since the added elements are fewer, this means all algorithms managed to add some groups of weighting vectors to the result.

Clustered data set W. In Figure 7, we use an identical setup with the previous experiment, only now the data set W is clustered. A clustered data set W simulates the case where user preferences are not independent, but there exist some groups of similar user preferences. The general observation is that both the execution time (Figure 7(a)) as well as the number of I/Os (Figure 7(b)) are smaller, when compared to the uniform data set W (see Figures 6(a) and 6(b)). Still, the relative performance between the different algorithms is similar as in the case of uniform data set W.

Correlated data set S. In Figure 8, we study the case of correlated data distribution for S. The results show that the branch-and-bound algorithms improve RTA in all setups both in terms of time and I/Os. This is quite important, because the CO data set is problematic for RTA even for small values of dimensionality. As shown in Figure 8(c), the reason for this behavior is the high cardinality $|RTOP_k|$ of the reverse top-k result in the case of CO data set. However, also in this case, the branch-and-bound algorithms significantly improve the performance of query processing. Notice that in the case of CO data set, the I/O cost is dominated by the I/Os induced on W, i.e., the bars in Figure 8(b) are completely colored, in contrast to the case of AC data set.

Varying k. Figure 9 shows the effect of increased values of k to time and I/Os for all algorithms. As shown in the chart, the branch-and-bound algorithms scale better than RTA as k increases.

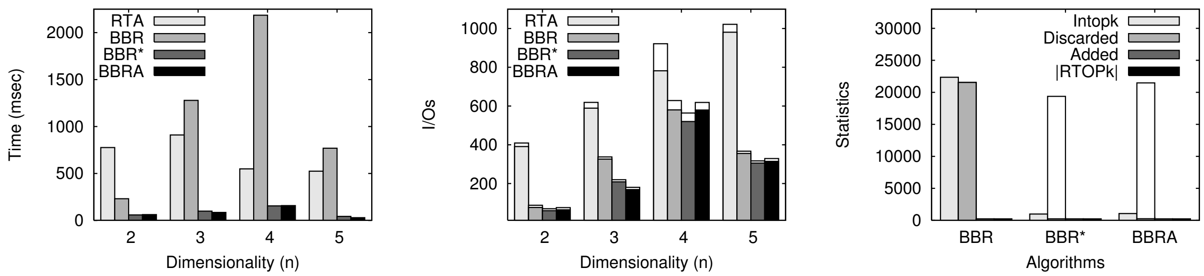
Clustered data sets S and W. In Figure 10, we employ a clustered data set both for S and W. Again, the branch-and-bound algorithms are superior to RTA in all setups. An interesting observation is that when both data sets are clustered, $BBR*$ and $BBRA$ manage to discard a very high number of MBRs (shown in Figure 10(c)), thus causing only few INTOPk invocations.

Anti-correlated data set S. In Figure 11, the performance of our algorithms is studied for anti-correlated data distribution of data set S for varying dimensionality. This distribution is quite demanding for all algorithms, therefore the results are depicted in log scale at the y-axis. In all cases $BBR*$ and $BBRA$ outperform RTA, thus verifying their superiority. Notice that we use log-scale on the y-axis of Figure 11(b), hence the white part of the stacked bars is the dominant cost.

Varying cardinality of S. Figure 12 shows the performance of all algorithms for increasing the cardinality of data set S. Notice that the branch-and-bound algorithms maintain their advantage over RTA as the size of S increases. Moreover, the performance of our algorithms is influenced by the cardinality of S only slightly, showing the scalability of our algorithms with respect to S. This is clearly shown by the number of induced I/Os, depicted in Figure 12(b).

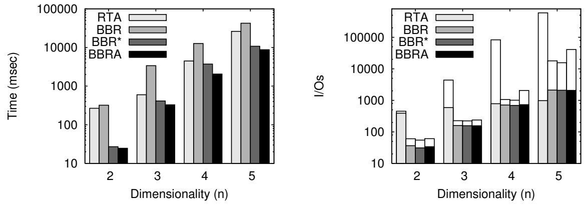
Varying cardinality of W. Figure 13 shows the performance of all algorithms for increasing the cardinality of data set W from 100K to 1M. When W increases in size, the performance of RTA drops significantly. In contrast, our algorithms $BBR*$ and $BBRA$ scale gracefully with $|W|$. This is a very strong result that demonstrates the efficiency of the branch-and-bound algorithms, in terms of pruning W.

Higher dimensions and skyline queries. In Figure 14, we conduct an experiment for high dimensions (6-9) to test the performance of all algorithms for stress conditions. Since in higher dimensionality the cardinality of the result set decreases, we generate the query workload by selecting queries out of the data set's skyline points. This means that on average each reverse top-k query will have more results, thus query processing becomes more expensive. As depicted in the charts, the performance of all algorithms dete-



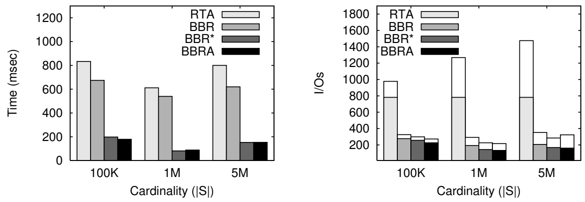
(a) Time. (b) I/Os. (c) Statistics.

Figure 10: Comparative performance of all algorithms for CL data sets S, W and varying dimensionality (n).



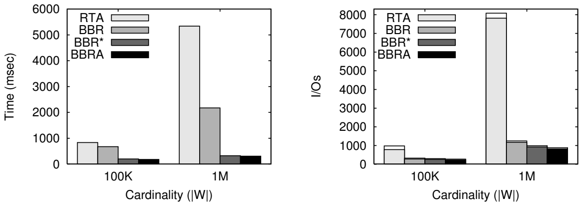
(a) Time. (b) I/Os.

Figure 11: AC data set and varying n.



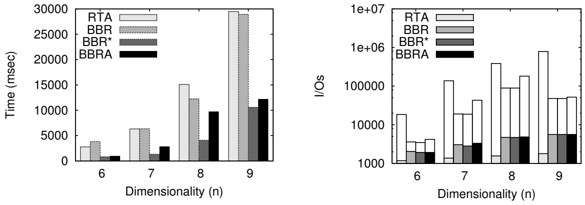
(a) Time. (b) I/Os.

Figure 12: Performance for varying $|S|$.



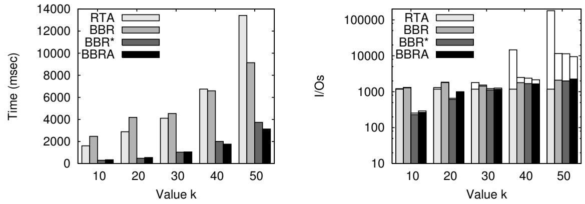
(a) Time. (b) I/Os.

Figure 13: Varying $|W|$.



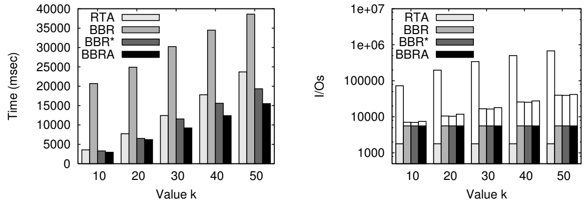
(a) Time. (b) I/Os.

Figure 14: Performance for high dimensions.



(a) Time. (b) I/Os.

Figure 15: Performance for HOUSE data set.



(a) Time. (b) I/Os.

Figure 16: Performance for COLOR data set.

riorates. However, $BBR*$ clearly outperforms all other algorithms in all setups consistently. Again, notice the use of log-scale in the y-axis of Figure 14(b).

Experiment with real data. Figure 15 shows the results from the first real data set (HOUSE) employed. Notice the log-scale in Figure 15(b). Also in these experiments, we use log-scale for the y-axis when reporting I/O. Clearly, $BBR*$ and $BBRA$ are more efficient than RTA, often by one order of magnitude. More importantly, the gain of our algorithms increases with the values of k. In Figure 16, we depict the results obtained in the case of the COLOR data set. Again, $BBR*$ and $BBRA$ outperform RTA, thus verifying their superiority also in the case of a demanding (9-dimensional) real data set. These results are in accordance with the study on the synthetic data. We conclude that our algorithms consistently improve the performance of RTA.

8. CONCLUSIONS

Reverse top-k queries constitute a useful tool for market analysis, since they help producers to identify those customers who are potentially interested in a particular product based on the customer preferences and the competitors' products. Currently, the state-of-the-art algorithm (RTA) needs to access each individual preference function and cannot add a preference function to the result set without evaluating the corresponding top-k query. In this paper, we propose a branch-and-bound algorithm for efficient reverse top-k query processing. Our algorithm alleviates the shortcomings of RTA by adding to the result set or discarding sets of preference functions instead of individual functions. To achieve this result, we study the conditions that ensure that a data point has a better rank than the query point for a set of preference functions. Based on these properties, we develop an efficient branch-and-bound algorithm and propose two optimizations that boost its performance. The experimental evaluation shows that our algorithm always outperforms RTA and performs efficiently in all cases.

9. ACKNOWLEDGMENTS

The work of Akrivi Vlachou was supported by the Action "Supporting Postdoctoral Researchers" of the Operational Program "Education and Lifelong Learning" (Action's Beneficiary: General Secretariat for Research and Technology), and is co-financed by the European Social Fund (ESF) and the Greek State. The research of Christos Doulkeridis was supported under the Marie-Curie IEF grant number 274063 with partial support from the Norwegian Research Council. The work of Yannis Kotidis has been co-financed by the European Union (European Social Fund – ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) - Research Funding Program: RECOST.

10. REFERENCES

[1] R. Akbarinia, E. Pacitti, and P. Valduriez. Best position algorithms for top-k queries. In *Proc. of Int. Conf. on Very Large Data Bases (VLDB)*, pages 495–506, 2007.

[2] A. Arvanitis, A. Deligiannakis, and Y. Vassiliou. Efficient influence-based processing of market research queries. In *Proc. of Conf. on Information and Knowledge Management (CIKM)*, pages 1193–1202, 2012.

[3] T. Bernecker, T. Emrich, H.-P. Kriegel, N. Mamoulis, M. Renz, S. Zhang, and A. Züfle. Inverse queries for multidimensional spaces. In *Proc. of Int. Symposium on Spatial and Temporal Databases (SSTD)*, pages 330–347, 2011.

[4] 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 Int. Conference on Management of Data (SIGMOD)*, pages 391–402, 2000.

[5] S. Chaudhuri and L. Gravano. Evaluating top-k selection queries. In *Proc. of Int. Conf. on Very Large Data Bases (VLDB)*, pages 397–410, 1999.

[6] E. Dellis and B. Seeger. Efficient computation of reverse skyline queries. In *Proc. of Int. Conf. on Very Large Data Bases (VLDB)*, pages 291–302, 2007.

[7] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *Proc. of Symposium on Principles of Database Systems (PODS)*, pages 102–113, 2001.

[8] S. Ge, L. H. U, N. Mamoulis, and D. W. Cheung. Efficient all top-k computation: A unified solution for all top-k, reverse top-k and top-m influential queries. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 25(5):1015–1027, 2013.

[9] U. Güntzer, W.-T. Balke, and W. Kießling. Optimizing multi-feature queries for image databases. In *Proc. of Int. Conf. on Very Large Data Bases (VLDB)*, pages 419–428, 2000.

[10] V. Hristidis, N. Koudas, and Y. Papakonstantinou. PREFER: a system for the efficient execution of multi-parametric ranked queries. In *Proc. of Int. Conference on Management of Data (SIGMOD)*, pages 259–270, 2001.

[11] F. Korn and S. Muthukrishnan. Influence sets based on reverse nearest neighbor queries. In *Proc. of Int. Conference on Management of Data (SIGMOD)*, pages 201–212, 2000.

[12] C. Li, B. C. Ooi, A. K. H. Tung, and S. Wang. DADA: a data cube for dominant relationship analysis. In *Proc. of Int. Conference on Management of Data (SIGMOD)*, pages 659–670, 2006.

[13] X. Lian and L. Chen. Monochromatic and bichromatic reverse skyline search over uncertain databases. In *Proc. of Int. Conference on Management of Data (SIGMOD)*, pages 213–226, 2008.

[14] A. Marian, N. Bruno, and L. Gravano. Evaluating top-*k* queries over web-accessible databases. *ACM Transactions on Database Systems*, 29(2):319–362, 2004.

[15] M. Miah, G. Das, V. Hristidis, and H. Mannila. Standing out in a crowd: Selecting attributes for maximum visibility. In *Proc. of Int. Conf. on Data Engineering (ICDE)*, pages 356–365, 2008.

[16] D. Papadias, P. Kalnis, J. Zhang, and Y. Tao. Efficient OLAP operations in spatial data warehouses. In *Proc. of Int. Symposium on Spatial and Temporal Databases (SSTD)*, pages 443–459, 2001.

[17] Y. Tao, V. Hristidis, D. Papadias, and Y. Papakonstantinou. Branch-and-bound processing of ranked queries. *Information Systems*, 32(3):424–445, 2007.

[18] Y. Tao, X. Xiao, and J. Pei. SUBSKY: efficient computation of skylines in subspaces. In *Proc. of Int. Conf. on Data Engineering (ICDE)*, page 65, 2006.

[19] A. Vlachou, C. Doulkeridis, Y. Kotidis, and K. Nørvåg. Reverse top-k queries. In *Proc. of Int. Conf. on Data Engineering (ICDE)*, 2010.

[20] A. Vlachou, C. Doulkeridis, Y. Kotidis, and K. Nørvåg. Monochromatic and bichromatic reverse top-k queries. *IEEE Transactions on Knowledge and Data Engineering*, 23(8):1215–1229, 2011.

[21] A. Vlachou, C. Doulkeridis, and K. Nørvåg. Monitoring reverse top-k queries over mobile devices. In *Proc. of Int. Workshop on Data Engineering for Wireless and Mobile Access (MobiDE)*, 2011.

[22] A. Vlachou, C. Doulkeridis, K. Nørvåg, and Y. Kotidis. Identifying the most influential data objects with reverse top-k queries. *PVLDB*, 3(1-2):364–372, 2010.

[23] Q. Wan, R. C.-W. Wong, I. F. Ilyas, M. T. Özsu, and Y. Peng. Creating competitive products. *PVLDB*, 2(1):898–909, 2009.

[24] D. Xin, C. Chen, and J. Han. Towards robust indexing for ranked queries. In *Proc. of Int. Conf. on Very Large Data Bases (VLDB)*, pages 235–246, 2006.

[25] B. Yao, F. Li, and P. Kumar. Reverse furthest neighbors in spatial databases. In *Proc. of Int. Conf. on Data Engineering (ICDE)*, 2009.

[26] K. Yi, H. Yu, J. Yang, G. Xia, and Y. Chen. Efficient maintenance of materialized top-k views. In *Proc. of Int. Conf. on Data Engineering (ICDE)*, pages 189–200, 2003.

[27] A. Yu, P. K. Agarwal, and J. Yang. Processing a large number of continuous preference top-*k* queries. In *Proc. of Int. Conference on Management of Data (SIGMOD)*, pages 397–408, 2012.