User-Centric Similarity Search

Konstantinos Georgoulas, Akrivi Vlachou, Christos Doulkeridis, and Yannis Kotidis

Abstract—User preferences play a significant role in market analysis. In the database literature there has been extensive work on query primitives, such as the well known top-k query that can be used for the ranking of products based on the preferences customers have expressed. Still, the fundamental operation that evaluates the similarity between products is typically done ignoring these preferences. Instead products are depicted in a feature space based on their attributes and similarity is computed via traditional distance metrics on that space. In this work we utilize the rankings of the products based on the opinions of their customers in order to map the products in a user-centric space where similarity calculations are performed. We identify important properties of this mapping that result in upper and lower similarity bounds, which in turn permit us to utilize conventional multidimensional indexes on the original product space in order to perform these user-centric similarity computations. We show how interesting similarity calculations that are motivated by the commonly used range and nearest neighbor queries can be performed efficiently, while pruning significant parts of the data set based on the bounds we derive on the user-centric similarity of products.

Index Terms—Similarity Estimation, Reverse top-k query, Top-k query, user preferences.

1 Introduction

Estimation of the similarity between objects is a fundamental operation in data management. For instance it is used to find pages or documents with similar words over the web [1] or in order to detect customers with abnormal behavior based on the products they buy [2]. Moreover, similarity computations can be performed for the detection of similar conversations and comments between users of the social networks (i.e. comments on Facebook, tweets on Twitter) [3].

Many different similarity metrics have been proposed for evaluating the similarity between two data items, such as the Euclidean distance and the cosine similarity. Such metrics suggest that the similarity between data items is computed based on their attributes, without taking into consideration users' opinions. For example, in business analysis the products are represented as points, defined by their attributes values. The closer two products are to each other according to the selected metric, the more similar they are.

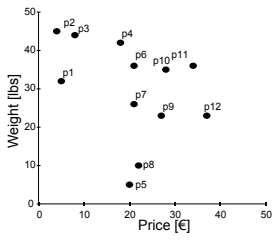
In our work we introduce a complementary user-centric approach for similarity computation, which takes into account users' preferences. For instance, a business manager would like to know the impact of its business products to customers, compared to their competitors existing products. It is quite important for her to know which of the products belong to the favorite list of as many different customers. This knowledge could be utilized to focus on products, which have similar groups of customers that rank them in high positions based on their preferences. Then, a more efficient marketing policy could be established, creating clusters of products that are preferable to specific customers.

- *K. Georgoulas and Y. Kotidis are with the Department of Informatics, Athens University of Economics and Business, 76 Patission Street, Athens GR 10434, Greece.*
  E-mail: {kgeorgou, kotidis}@aueb.gr.
- *A. Vlachou is with Institute for the Management of Information Systems, Research Center "Athena", Artemidos 6 & Epidavrou Street, Marousi GR 15125, Greece.*
  E-mail: vlachou@aueb.gr.
- *C. Doulkeridis is with the Department of Digital Systems, University of Piraeus, 80, M. Karaoli & A. Dimitriou Street, Piraeus GR 18534,Greece.*
  E-mail: cdoulk@unipi.gr.

Thus, a business manager should be able to perform a query that returns products (even products that have never been rated) which are similar based not only on their characteristics but also on the users' preferences. She needs to see the data through the eyes of the users while performing similarity computations based on the available users' preferences. In our framework, users' preferences are expressed as vectors of weighting factors for items' attributes. In order to perform such kind of similarity computations, we exploit a query type, termed a reverse top-k query [4]. In contrast to a top-k query that returns the k products with the best score for a specific customer, the result of a reverse top-k query is the set of customers for whom a given product belongs to their top-k set. Our work is also applicable in case products have recently launched in the market or products being in the designing phase of the manufacturing process and there have been no expressed users opinions for them yet.

Example 1. *Consider an example from business analysis, where we would like to estimate the similarity between products based on the preferences their customers have expressed for them. A common way to rank products for a customer is to execute a top-k query that assigns scores to each product. In a typical setting, the top-k query utilizes a weight $w[i]$ for each attribute $p[i]$ of product p and combines these weights and attributes via a scoring function f. The particular set of weights are different for each customer. A score function often used in the literature [4] is the linear function f_w, which for a product p is computed as: $f_w(p) = \sum w[i] * p[i]$. Without loss of generality we assume that lower scores are better.*

The weights of a customer define a vector w that is of the same dimensionality as the vectors used to represent the products. They can thus be presented in the same d-dimensional feature space, where d is the number of products' attributes. In Figure 1 we can see twelve products $p_1, \ldots, p_{12}$ and the weighting vectors $w_1, \ldots, w_6$ of six different customers. The dimensionality of this space is d=2. If we utilize the Euclidean distance for computing the similarity between products, then it is evident that product p_9 is more similar to products p_7 and p_{12}. A natural question is whether this similarity evaluation is natural for the users, based on their



Product	Price [€]	Weight [lbs]
p_1	5	32
p_2	4	45
p_3	8	44
p_4	18	42
p_5	20	5
p_6	21	36
p_7	21	26
p_8	22	10
p_9	27	23
p_{10}	28	35
p_{11}	34	36
p_{12}	37	23

Product	*Customers* that include *Product* in their favorite list
p_1	$\{w_3, w_4, w_6\}$
p_2	$\{w_3, w_4\}$
p_3	$\{w_3\}$
p_5	$\{w_1, w_2, w_4, w_5, w_6\}$
p_8	$\{w_1, w_2, w_5, w_6\}$
p_9	$\{w_1, w_2, w_5\}$

User	w[Price]	w[Weight]
w_1	0.15	0.85
w_2	0.07	0.93
w_3	0.93	0.07
w_4	0.73	0.27
w_5	0.24	0.76
w_6	0.52	0.48

Fig. 1. Example of products, user preferences and reverse top-k sets.

expressed preferences, depicted in the six weighting vectors?

Without getting in the details of the top-k/reverse top-k queries (these will be presented in the next sections), product p_8 is in the favorite list of four customers (w_1, w_2, w_5, w_6) while three of these customers and more specific customers with weighting vectors w_1, w_2, w_5 include product p_9 in their favorite list of products too. On the other hand, neither p_7 nor p_{12} are included in the favorite lists of any one of the six customers. Thus, in the perspective of the specific set of the six customers and in more detail for customers with weighting vectors w_1, w_2, w_5 products p_8 and p_9 are considered similar while there is no apparent similarity between p_9 and p_7, p_{12} according to their preferences. Put differently, the reverse top-k set (for k=3) of products p_9 and p_8 (i.e. the set of customers that have in their top-3 favorite list product p_9 and, respectively, product p_8) include customers with weighting vectors w_1, w_2, w_5. Thus, the coexistence of the same customers in the reverse top-k sets of two or more products is a positive testimony for their similarity. This simple example shows that the similarity between products can be quite different if we take user preferences into account, than looking at the products as individual points in their feature-space, isolated from the effects of these user preferences.

In our work, we utilize the Jaccard coefficient to perform similarity computations between the resulting sets of the reverse top-k queries. An extended notion of similarity that further takes into account the ranks of the products is also considered. We complement these new similarity metrics with two query types we introduce, termed θ-similarity and m-nearest neighbor queries that are analogous to the well known range and nearest neighbor queries, but differ in that they evaluate the similarity between products by looking at their reverse top-k sets. Unfortunately, reverse top-k queries are known to incur high computational cost, meaning that a brute-force evaluation of these queries is impractical, even for moderate data sets. In this work we identify important properties of the reverse top-k results that allow us to compute upper and lower bounds on the user-centric similarity between products. Per these bounds, we are able to execute these queries efficiently utilizing a conventional R-tree index on the product space. We further discuss additional optimizations that re-use previously computed reverse top-k sets in order to further bound the similarity of products under investigation.

In summary, this paper makes the following contributions:

- We introduce a novel framework for *user-centric* similarity search, which capitalizes on rankings of products based on user preferences to discover similar products.
- We define two novel query types (θ-similarity and m-nearest neighbor) for user-centric similarity search, identify effective score bounds (Section 3), and present efficient query processing algorithms that prune the search space by exploiting the derived bounds and traditional index structures (Section 4).
- We explain how our techniques can be extended when a different similarity metric is employed, which captures user-centric similarity in a more fine-grained manner (Section 5).
- We show that results computed while a query is being processed can be exploited to derive more tight bounds, thus greatly improving the performance of query processing (Section 6).
- We perform a detailed experimental evaluation that demonstrates both the efficiency and effectiveness of user-centric similarity search (Section 7). A key finding of our experiments is that by exploiting user preferences, we often obtain very different results compared to traditional similarity computations. In two user-studies that we contacted on a real dataset, we found that in the majority of the queries, the users preferred the results obtained via our techniques.

In addition, we present definitions that help set up the similarity framework (Section 2), we discuss related work (Section 8), and we provide concluding remarks (Section 9).

2 PRELIMINARIES

In this section, we present definitions and preliminaries that we utilize to better introduce and describe our techniques.

Given a data set P of products with d attributes, we define each product as $p = \{p[1], p[2], \ldots, p[d]\}$ where $p[i]$ is the value of the $i\text{-}th$ attribute. For a set C of customers with cardinality n, a customer $c \in C$ describes her preferences for the products' attributes by a weighting vector $w = \{w[1], w[2], \ldots, w[d]\}$, where weight $w[i]$ denotes this customer's preference for the $i\text{-}th$ attribute of every product p. We assume that both values of attributes and customer's preferences are positive real numbers and without loss of generality we assume that $\sum_{i=1}^{d} w[i] = 1$.

In order to perform both top-k and reverse top-k queries, a score function is required that ranks the products based on the values of attributes $p[i]$ and the customer's preferences $w[i]$. The weighted sum function, we referred to in the previous section, is known as linear and assigns a score $f_w(p) = \sum_{i=1}^{d} w[i] * p[i]$ to product p for this customer.

A top-k query returns the k data items with the best scores[1] and can be expressed as $TOP_k(w)$, where w is a d-dimensional vector $w = \{w[1], ...w[d]\}$ that represents preference values.

Definition 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 the set of products such that $TOP_k(w) \subset P, |TOP_k(w)| = k$ and $\forall p_1, p_2 : p_1 \in TOP_k(w), p_2 \in P - TOP_k(w)$ it holds $f_w(p_1) \leq f_w(p_2)$*

Definition 2 (Dominance)**.** *Given two points p, q in $\mathbb{R}^d$ space, p dominates q, $p \prec q$, if $\forall i$ with $i = 1 \ldots d, i \neq j$ is $p[i] \leq q[i]$ and $\exists$ at least one j for which $p[j] < q[j]$.*

1. Without loss of generality, in this paper, lower scores are preferable in top-k queries.

We also use $p \preceq q$ to denote the case where either p dominates q, or p and q coincide.

The dominance relationship between any two points p and q (with $p \prec q$) implies that for any *increasing monotone* function f it holds that $f(p) < f(q)$. Any score function that satisfies the aforementioned property could be used for ranking the products in our framework. In particular, the linear weighting function f_w is monotonically increasing. The following lemma captures this property and indicates that a point that dominates another point also has a better rank for *any* linear weighting function f_w.

Lemma 1. *Given two points p and q where p dominates or coincides with q ($p \preceq q$), it holds that for any linear weighting function f_w: $f_w(p) \leq f_w(q)$.*

Moreover, in order to perform the similarity computations of our example we should execute reverse top-k queries, which return the weighting vectors of every customer for whom the product p, which is given as input to the query, belongs to hers k-first top ranking products. In contrast to a top-k query that takes as an input a specific customer and is performed over the products data set, a reverse top-k query receives as input a product and is executed over the set of all users' weighting vectors.

Definition 3 (Reverse top-k query)**.** *Given a product q and a positive number k, as well as two data sets P and W of products 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 $f_{w_i}(q) \leq f_{w_i}(p)$.*

A key property between the reverse top-k result sets of two products is based on the concept of point dominance, as described in the following lemma.

Lemma 2. *Given two points p and q where p dominates q ($p \prec q$), it holds that $RTOP_k(q) \subseteq RTOP_k(p)$.*

Given the $RTOP_k$ results of every product in the data set, we can evaluate the similarity between two products in a user-centric manner by looking at their reverse top-k resulting sets. In our motivational example, a possible similarity query would be: *"Which products have $RTOP_k$ results that are more than 80% similar to the $RTOP_k$ set of a given product p_i"*. Of course, the answer to this query involves the notion of similarity of the $RTOP_k$ sets, that we have to define. Moreover, it suggests the execution of reverse top-k queries for all the products of the data set, something that is not applicable to large data sets. Another interesting query that raises similar challenges is to execute Nearest Neighbor searches over the products by considering their $RTOP_k$ results. In what follows we introduce an intuitive similarity metric called the Jaccard coefficient that we exploit for the evaluation of the similarity computations, we formally define our queries and then discuss techniques for the efficient execution of them in large datasets.

3 Similarity & Queries

In this paper we introduce a user-centric approach for similarity computation, in which the similarity of two products is defined by taking into account the user preferences. Two products are considered to be similar if they satisfy the same user preferences, which in turn means that their reverse top-k sets are similar.

More formally, suppose a set of customers C. Each customer c_i has defined hers d preferences with a weighting vector w_i. As already explained, given a set of products P, a weighting vector w_i would be part of the result of the reverse top-k set of a product q if the query point q is one of the k products that the top-k query for customer c_i returns.

In order to perform similarity computation over the products' $RTOP_k$ sets we introduce a metric for the distance (i.e. similarity) of these sets. A well known similarity function for sets of objects is the Jaccard coefficient. The Jaccard coefficient measures similarity between two sets as a fraction of the size of their intersection divided by the size of the union of the two sets.

Definition 4 (User-centric Similarity)**.** *Given two products p_i and p_j, the similarity $sim(p_i, p_j)$ between p_i, p_j is computed as the Jaccard coefficient of their corresponding $RTOP_k$ sets: $sim(p_i, p_j) = Jaccard(RTOP_k(p_i), RTOP_k(p_j)) = \frac{|RTOP_k(p_i) \cap RTOP_k(p_j)|}{|RTOP_k(p_i) \cup RTOP_k(p_j)|}$*

As an example, Figure 1 depicts the non-empty reverse top-k sets for all existing products, for $k = 3$. Let us consider the reverse top-k sets for p_1 and p_2. The intersection of the two sets is of size 2, while their union of size 3. Thus, the user-centric similarity of the corresponding products is $sim(p_1, p_2) = \frac{2}{3}$. Also, notice that based on user-centric similarity $sim(p_1, p_3) = sim(p_1, p_5) = \frac{1}{3}$, which is quite different than when the Euclidean distance is used, according to which p_1 is much closer to p_3 than p_5.

In our framework we consider two query types that are analogous to the commonly used range and Nearest Neighbor queries in the Euclidean space.

Definition 5 (θ-$similarity(q)$ query)**.** *Given a query product q and threshold θ such that $\theta \in [0, 1]$, the result set of the θ-$similarity$ query is the set of products such that $sim(p, q) \geq \theta$.*

Definition 6 (m-$NN(q)$ query)**.** *Given a query product q and a positive integer m, the result set of the m-$NN(q)$ is a set of m products such that $\forall p_1, p_2 : p_1 \in m\text{-}NN(q), p_2 \in P - m\text{-}NN(q)$ it holds $sim(p_1, q) \geq sim(p_2, q)$.*

Continuing our example depicted in Figure 1 and assuming that $q = p_1$ then p_2 is closer to q than p_3, while the inverse holds for the Euclidean distance. Also, based on user-centric similarity, p_8 is more similar to q than p_4, namely p_8 is the 3-NN. In contrast, using Euclidean distance, p_4 is the 3-NN of q, while p_8 is the 8-NN.

In the discussion that follows we will (for brevity) represent (virtually) each product's p_i reverse top-k result as a binary vector r_{p_i} of size n, where n is the size of C. Bit j of vector r_{p_i} is set if the j-th weighting vector w_j, which represents the preferences of the customer c_j, belongs to the reverse top-k set of p_i, otherwise is clear. Since r_p is a vector-representation of the corresponding set, we will also denote the similarity computation as $sim(p_i, p_j) = \frac{|r_{p_i} \cap r_{p_j}|}{|r_{p_i} \cup r_{p_j}|}$ using the vectors instead of the corresponding sets.

4 Query Processing

In this section, we describe the indexing structure used for efficient user-centric similarity search, we define upper and lower bounds on the similarity of query point q to index entries, and we design the similarity search algorithms.

4.1 Indexing

To facilitate efficient access to the product data set, a multidimensional access method is required. In this work, we employ

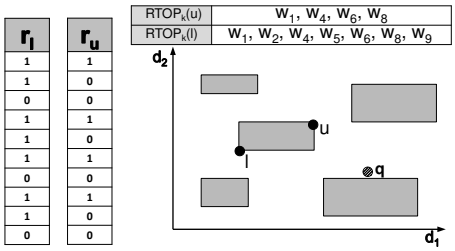


Fig. 2. $RTOP_k$ result sets of MBR's $M(l, u)$ corners.

the R-Tree [5], which is a popular multidimensional index widely used for similarity search. The R-Tree organizes the products in (hyper)rectangles called MBRs (Minimum Bounding Rectangles) that form a tree hierarchy. At the lowest (leaf) level, each MBR is defined as the minimum-sized rectangle that contains/encloses a set of products, whereas at higher levels in the tree multiple MBRs from the next level are grouped together and represented by a new MBR that includes all the products of the group's rectangles. Each MBR $M(l, u)$ is defined by two points, namely its lower corner l and upper corner u, which have in each dimension the minimum and maximum coordinates, respectively, among all points of the MBR. Obviously, in the general case, the points l and u may not be part of the data set, i.e., they may not correspond to actual products.

In our work, we enhance the MBRs of the R-tree with additional information, which consists of the reverse top-k sets $RTOP_k(l)$ and $RTOP_k(u)$ of the lower and upper corner, respectively. To illustrate this better, consider the example of Figure 2, where for a hypothetical set of objects and a group of users given the $RTOP_k(l)$, $RTOP_k(u)$ we construct the virtual vectors r_l and r_u of an MBR in the way that we previously described. The corners l and u have four common weighting vectors(w_1, w_4, w_6, w_8) in their $RTOP_k$ sets. As a result, both vectors r_l and r_u have a value of one to first, fourth, sixth and eighth coordinate. However, there are many other $w_i's$ that belong to one of the two products $RTOP_k$ sets but not to the other (e.g., w_2).

Interestingly, as will be shown in the following, we can exploit the binary vectors r_l and r_u of an MBR to bound the similarity between a query point q and any product p contained in the MBR, without computing the reverse top-k result of p.

4.2 Similarity Bounds

Given a query point q and an MBR $M(l, u)$ defined by corners l and u, we define upper and lower bounds on the similarity $sim(p, q)$ of q with any product p enclosed in this MBR (r_q is a binary vector with the reverse top-k result of point q).

Lemma 3. *The upper bound of the Jaccard similarity of any point p enclosed by an MBR $M(l, u)$ with query point q is:* $max\_sim(M, q) = \frac{|r_q \cap r_l|}{|r_q \cup r_u|}$.

Proof. For any point p enclosed in MBR $M(l, u)$ it holds that $l \preceq p$ and $p \preceq u$. Based on the key property described in Lemma 2, this means that $r_p \subseteq r_l$ and $r_u \subseteq r_p$. It follows that $r_q \cap r_p \subseteq r_q \cap r_l$ and $r_q \cup r_u \subseteq r_q \cup r_p$ respectively. Then, $sim(p, q) = \frac{|r_q \cap r_p|}{|r_q \cup r_p|} \leq \frac{|r_q \cap r_l|}{|r_q \cup r_p|} \leq \frac{|r_q \cap r_l|}{|r_q \cup r_u|} = max\_sim(M, q)$. □

Lemma 4. *The lower bound of the Jaccard similarity of any point p enclosed by an MBR $M(l, u)$ with query point q is:* $min\_sim(M, q) = \frac{|r_u \cap r_q|}{|r_l \cup r_q|}$.

Proof. Similar to the above. □

We will exploit these bounds to design efficient algorithms for user-centric similarity search, as will be shown next.

4.3 Algorithms

In what follows, we describe query processing algorithms for the evaluation of the two query types introduced in Section 3. For both queries, we assume that the set of products is indexed using an R-tree extended to contain the $RTOP_k$ sets of the lower and upper corner of each MBR $M(l, u)$. For efficient computation of the $RTOP_k$ sets, we utilize the RTA algorithm [4]. RTA expedites query processing by discarding weighting vectors that cannot belong to the result set of $RTOP_k(q)$, without evaluating the corresponding top-k queries. As such, RTA reduces the number of top-k evaluations, based on the observation that top-k queries defined by similar weighting vectors return similar results [6]. For this reason, prior to query processing, the weighting vectors are sorted based on their pairwise similarity. By exploiting this order, RTA avoids processing top-k queries that correspond to subsequent vectors in this order. During processing, a buffer containing the k objects (results) of the previous top-k query that was processed is used to discard subsequent vectors, if the objects in the buffer are ranked higher than q.

RTA is not significantly affected when the cardinality of the product space or the value of k increases [4]. On the other hand, in case of increase of the weighting vectors cardinality, the number of top-k evaluations necessary for the computation of the result set of a $RTOP_k$ query also increase. In the worst case, the RTA algorithm has to process $|W|$ top-k queries, but in practice the algorithm returns the correct result by evaluating much fewer top-k queries than $|W|$. A lower bound of at least $|RTOPk(q)|$ top-k queries should be executed, since no weighting vector can be added in the result set without evaluating the respective top-k query.

4.3.1 $\theta-$similarity Queries

By definition, the θ-$similarity(q)$ query retrieves all products p that satisfy the condition $sim(p, q) \geq \theta$. We will answer this query utilizing the R-tree index by recursively expanding its nodes in a top-down fashion, where a node is expanded only if it may contain a candidate product for inclusion in the result set.

Algorithm 1 describes the pseudo code. A priority queue L is used to traverse the R-tree recursively and L is initialized by the root of the R-tree. The products that belong to the result set are maintained in RES. During the search, in each iteration a node is examined, and if it holds that $min\_sim(M, q) \geq \theta$, then all nodes that belong to this MBR (the subtree of the current node) are definitely part of the query result set. Thus, we recursively visit the subtree and add all products indexed by it to the result set (lines 2-4). In case the next node M is a leaf node, then the Jaccard similarity of the indexed products to q has to be computed. Therefore, for each indexed product a reverse top-k query is executed, and if $sim(p, q) \geq \theta$ the corresponding product is added to the result set (lines 6-11). Finally, if M is a non-leaf node, then M is expanded and its child nodes M_j are inserted into L. Nodes M_j for which it holds that $max\_sim(M_j, q) < \theta$ can

Algorithm 1 Theta_Query(q, L, θ, RES)

Input: q is the query point
L is a priority queue
θ is the query parameter
RES is the result set

1: $M = L$.dequeue()
2: **if** $min\_sim(M, q) \geq \theta$ **then**
3: $RES = RES \cup p, \forall p \in subtree(M)$
4: **end if**
5: **if** M.type=LEAF **then**
6: **for** $\forall p_i \in M$ **do**
7: $r_{p_i} = executeRTOP_k(p_i)$
8: **if** $sim(p_i, q) \geq \theta$ **then**
9: $RES = RES \cup p_i$
10: **end if**
11: **end for**
12: **else**
13: **for** $\forall M_j \in M$ **do**
14: **if** $max\_sim(M_j, q) \geq \theta$ **then**
15: L.enqueue(M_j)
16: **end if**
17: **end for**
18: **end if**
19: **if** L is not empty **then**
20: $Theta\_Query(q, L, \theta, RES)$
21: **end if**

Algorithm 2 Nearest_Neighbor(q, L, m, nn)

Input: q is the query point
L is a priority queue
m is the number of Nearest Neighbors
nn is the list of Nearest Neighbors

1: $M = L$.dequeue()
2: **if** M.type=PRODUCT **then**
3: nn.add(M)
4: **end if**
5: **if** nn.size$== m$ **then**
6: return nn
7: **end if**
8: **if** M.type=LEAF **then**
9: **for** $\forall p_i \in M$ **do**
10: $r_{p_i} = executeRTOP_k(p_i)$
11: L.enqueue(p_i)
12: **end for**
13: **else**
14: **for** $\forall M_j \in M$ **do**
15: L.enqueue(M_j) {L is maintained as a priority queue based on the upper bounds $max\_sim(M_j, q)$}
16: **end for**
17: **end if**
18: **if** L is not empty **then**
19: $Nearest\_Neighbor(q, L, m, nn)$
20: **end if**

be safely discarded. Thus, only the nodes M_j that may potentially contain products that belong to the result set, based on the r vectors of its MBR corners are inserted into L (lines 14-16), while the remaining nodes are discarded. In the worst case, where there is no pruning, the algorithm performs $|P|$ $RTOP_k$ evaluations. However, in the average case only a fraction of $|P|$ queries has a non-empty $RTOP_k$ set. A smaller value of k value results in more empty $RTOP_k$ results sets that would be pruned by our algorithm. In practice, the algorithm's performance is dominated by the complexity of $RTOP_k$ query's execution algorithm (RTA), which we have already described in detail in the previous section.

4.3.2 Nearest Neighbor Queries

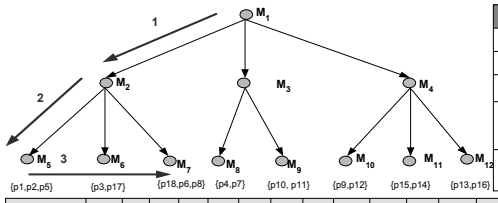
Recall that the m-$NN(q)$ query seeks to find the m most similar products to a specific product q. In order to process the m-NN query, we utilize an algorithm that performs a depth-first traversal of the R-tree.

Algorithm 2 describes the pseudo code. In order to find the m nearest neighbors of a given product q, our algorithm traverses the R-tree, beginning from the root node, and searches the m products $p_{nn_1} \dots p_{nn_m}$ for which $sim(p_{nn_i}, q)$ is maximized. A priority queue L is used to provide access to the MBRs M_j in descending order of their upper bound of similarity ($max\_sim(M_j, q)$). The priority queue is initialized by the root of the R-tree. As long as the m nearest neighbor have not been found and L is not empty, the algorithm calls itself recursively (line 19). Each time the first element M of L is removed. If it is a single product then it is added to the list of nearest neighbors and the list is returned when its size equals to m (line 6). The sorted access and the properties of the upper bounds ensure us that nn list's element are the m nearest neighbors as no other product can have a higher similarity. When a leaf node M of the R-tree is reached (line 8), a reverse top-k query is executed for every product p_i, that belongs to the specific MBR M and the r_{p_i} vector is populated (line 10). The products are also inserted to the priority queue L and their upper bound is equal to their actual similarity to the query product $sim(p_i, q)$. Finally, if the next node M is a non-leaf node, then M is expanded and all MBRs M_j that are enclosed in M are inserted to the priority queue L (lines 14-16).

An important feature of this algorithm is the sorted access employed. In more detail, pruning is achieved because our algorithm ensures that nodes M for which it holds that the upper bound is smaller than the similarity of the m-th nearest neighbor p_{nn_m} (i.e., $max\_sim(M_j, q) < sim(p_{nn_m}, q)$) are never removed from L. The entire sub tree of M will not be processed further, thus also avoiding the overhead of processing its leaf-level MBRs and their enclosed points. Perhaps more importantly, the respective reverse top-k queries for the enclosed points are avoided. Also, we could further increase pruning by maintaining an extra priority queue L' of size m, that stores, in reverse order, the m higher similarity values between the products that have already enqueued in L and the query point. As a consequence, products or MBRs that they would later be examined by algorithm, which similarity (i.e. upper bound in case of MBR) with q is smaller than the value of the first element of L' can be pruned and do not enqueued in L reducing algorithm's storage cost.

In terms of complexity, as in the case of the θ-similarity query's algorithm, the presented algorithm would perform $|P|$ $RTOP_k$ queries in the worst case, but as shown in our experimental study the number of $RTOP_k$ queries is much smaller in practice, because of the effective pruning we utilize.

Example 2. *As an example, we assume the R-tree depicted in Figure 3 and a 1-NN query for a given point q. After expanding the root node M_1 of the R-tree (step 1), its children MBRs, namely M_2, M_3 and M_4, will be inserted in L (step 2). Then, M_2 is extracted from the top of the queue and its children nodes M_5, M_6 and M_7 are added (step 3). The algorithm continues with the next MBR at the top of the queue, which is M_5. This is a leaf node and contains three products (p_1, p_2, p_5), their similarity with query point q is computed and the products are inserted in L (step 4). In our example we assume that p_1 is the product with the highest*



Contents of Queue L	
Step1	M_1
Step2	M_2, M_3, M_4
Step3	M_5, M_7, M_3, M_6, M_4
Step4	M_7, p1, M_3, p2, p5, M_6, M_4
Step5	p1, M_3, p2, p5, M_6, p8, p6, M_4, p18

	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}	M_{11}	M_{12}	p1	p2	p5	p6	p8	p18
Maximum bound	0.92	0.9	0.7	0.4	0.9	0.6	0.8	0.7	0.64	0.4	0.3	0.35	0.75	0.65	0.62	0.5	0.55	0.4
Minimum bound	0.6	0.6	0.5	0.3	0.5	0.4	0.4	0.6	0.5	0.3	0.2	0.3						

Fig. 3. NN algorithm's R-tree traversal.

similarity, namely $sim(p_1, q) = 0.75$. *The algorithm continues with the next top item in* L *(*M_7*). The similarity of the products in* M_7 *with* q *is computed and they are enqueued in* L *(step 5). Given that these products have a smaller similarity than* p_1 *(see table of fig 3), the algorithm terminates after* p_1 *is removed from* L *and product* p_1 *is returned as the nearest neighbor to* q.

5 Extended Jaccard Similarity

The result set of a top-k query is bounded to the k products with the best score, while the reverse top-k result set cannot be bounded; it can (in theory) become equal to the number of customers or it can even be empty. Put differently, a reverse top-k query could possibly return no weighting vector (customer) for a given product. In the case of products with small or empty reverse top-k sets, the similarity estimation could be insufficient or impossible. A workaround to this problem would be to increase the value of parameter k enlarging this way the size of the reverse top-k sets. However, this could lead to inaccurate estimation of similarity, caused by the fact that the Jaccard coefficient does not take into account the actual ranking of the product in the top-k list of a customer, who belongs to the product's $RTOP_k$ set. Thus, the Jaccard coefficient may consider two products similar, even if the customers rank the first one in high positions and the second one in much lower positions. Such discrepancies become more evident, when the value of parameter k increases.

In order to alleviate this limitation of the Jaccard coefficient, we propose an extended schema that takes into account the actual ranking position of a product in the top-k set of a customer. In a nutshell, instead of assigning one as the value of coordinate $r_p[i]$, in case customer c_i belongs to the reverse top-k set of product p, we assign a positive value that is derived from the ranking position of the product in the customer's list. This results in vectors r_p consisting of real positive numbers instead of binary values. In this way, two products that are ranked in similar positions by a group of customers are considered more similar by extended Jaccard than any other product which ranking is quite different by the same customers.

Let $rank_i(p)$ denote the ranking of product p in the top-k list of customer c_i. It holds that $1 \leq rank_i(p) \leq k$. The coordinates of vector r_p are non-negative numbers derived from the following formula:

$$r_p[i] = \begin{cases} \frac{1}{rank_i(p)}, & \text{if } p \text{ belongs in the top-}k \text{ list} \\ & \text{of customer } c_i \\ 0, & \text{otherwise} \end{cases}$$

where each coordinate is defined as the inverse of a product's rank. Obviously, without loss of generality, any other non-negative, decreasing function can be used instead of the inverse function, as long as it implies that products ranked higher in the list (i.e., smaller score) will be assigned larger values.

The original Jaccard coefficient cannot be used to compute the similarity between non-binary vectors. We thus utilize the *extended Jaccard coefficient*, which is applicable to any kind of real vectors A, B and computes their similarity by the following formula ($\cdot$ stands for the inner-product operator):

$$extJaccard(A, B) = \frac{A \cdot B}{\|A^2\| + \|B^2\| - A \cdot B}$$

In our setting, we use the extended Jaccard coefficient to denote the similarity of two products, based on the derived vectors of two products p_i and p_j: $ext\_sim(p_i, p_j) = extJaccard(r_{p_i}, r_{p_j})$

Having defined the similarity between products, what remains is a way to derive upper and lower bounds for their similarity. We suggest that the bounds for the similarity $ext\_sim(p, q)$, where p is any product that belongs to a specific MBR $M(l, u)$ and q is the query point, are given by the following formulas:

$$ext\_max\_sim(M, q) = \frac{2 * r_l \cdot r_q}{r_u^2 + r_q^2}$$

$$ext\_min\_sim(M, q) = \frac{r_u \cdot r_q}{r_l^2 + r_q^2}$$

where r_u and r_l are the vectors for corners u and l of a specific MBR $M(l, u)$ respectively, which are defined similarly to r_p.

In the proofs presented below, we show the correctness of the formulas for the similarity bounds. Both these proofs rely on Lemma 1.

Lemma 5. *The upper bound of the extended Jaccard similarity of any point* p *enclosed by an MBR* $M(l, u)$ *with query point* q *is:* $ext\_max\_sim(M, q) = \frac{2*r_l \cdot r_q}{r_u^2 + r_q^2}$.

Proof: We prove that: $extJaccard(r_q, r_p) \leq \frac{2*r_l \cdot r_q}{r_u^2 + r_q^2}$ We know that

$$extJaccard(r_q, r_p) = \frac{r_p \cdot r_q}{r_p^2 + r_q^2 - r_q \cdot r_p} \leq 1$$

Adding the same positive amount (specifically $r_p \cdot r_q \geq 0$) in both the denominator and the numerator of the fraction above, whose value is lower than one and both the denominator and the numerator are positive numbers, its value is getting greater. More formally, suppose that $\frac{a}{b} < 1$ and, thus, $a < b$ for $a, b > 0$. It holds that $ac < bc$, $\forall c > 0$ and, similarly, $ab + ac < ab + bc$. This means that $a(b + c) < b(a + c)$ and, thus, $\frac{a}{b} < \frac{a+c}{b+c}$.

Thus, the following inequality holds.

$$\begin{aligned} extJaccard(r_q, r_p) &\leq \frac{r_p \cdot r_q + r_p \cdot r_q}{r_p^2 + r_q^2 - r_p \cdot r_q + r_p \cdot r_q} \\ &= \frac{2 * r_p \cdot r_q}{r_p^2 + r_q^2} \end{aligned}$$

Since $l \preceq p$ and $p \preceq u$, then based on Lemma 1 it holds that for any linear weighting function f_w: $f_w(l) \leq f_w(p) \leq f_w(u)$, which means that for any customer c_j: $rank_j(u) \geq rank_j(p) \geq rank_j(l)$, and equivalently $\forall i : r_u[i] \leq r_p[i] \leq r_l[i]$. Thus,

we replace r_p with r_u, r_l in the denominator and in the numerator respectively. Thus, $extJaccard(r_q, r_p) \leq \frac{2*r_l \cdot r_q}{r_u^2+r_q^2} = ext\_max\_sim(M, q)$ □

Lemma 6. *The lower bound of the extended Jaccard similarity of any point p enclosed by an MBR $M(l, u)$ with query point q is:* $ext\_min\_sim(M, q) = \frac{r_u \cdot r_q}{r_l^2+r_q^2}$.

Proof: We prove that: $extJaccard(r_q, r_p) \geq \frac{r_u \cdot r_q}{r_l^2+r_q^2}$ It holds that

$$extJaccard(r_q, r_p) = \frac{r_p \cdot r_q}{r_p^2 + r_q^2 - r_q \cdot r_p} \geq \frac{r_p \cdot r_q}{r_p^2 + r_q^2}$$

Since $l \preceq p$ and $p \preceq u$, then based on Lemma 1 it holds that for any linear weighting function f_w: $f_w(l) \leq f_w(p) \leq f_w(u)$, which means that for any customer c_j: $rank_j(u) \geq rank_j(p) \geq rank_j(l)$, and equivalently $\forall i : r_u[i] \leq r_p[i] \leq r_l[i]$. Thus, we replace r_p with r_u, r_l in the denominator and in the numerator respectively would be: $extJaccard(r_q, r_p) \geq \frac{r_l \cdot r_q}{r_u^2+r_q^2} = ext\_min\_sim(M, q)$ □

6 OPTIMIZATIONS

A potential drawback of the previously presented algorithms, is that each time a leaf-level MBR M is visited, a reverse top-k query needs to be processed *for each* product enclosed in M. This induced cost is a substantial contributor to the overall processing cost of the algorithms. In this section we present optimizations that seek to minimize the number of reverse top-k computations resulting in faster query processing.

In order to minimize the number of reverse top-k evaluations, we exploit the result sets of already evaluated reverse top-k queries in order to obtain tighter upper and lower similarity bounds for (any product p enclosed by) a leaf MBR M and query point q. By utilizing the updated bounds, we manage to prune many points of a given leaf MBR without computing their reverse top-k sets, as soon as we are certain that they cannot be part of the query result.

More technically, every time a $RTOP_k(p_i)$, query is performed, where p_i is a product enclosed by a leaf MBR M, its result set r_{p_i} is maintained in an in-memory R-tree I. Before computing the reverse top-k query of a newly visited product p, we first retrieve a carefully selected subset of vectors r_{p_i} from I, and exploit them to update the bounds (details will be given in what follows) for the value of $sim(p, q)$ (or $ext\_sim(p, q)$). In the case of a θ-similarity query, we can discard p without performing a reverse top-k query for it, in case the updated upper bound is lower than the defined threshold. In the case of a m-NN query, the updated upper bound may push p further back in the priority queue, postponing the computation of its reverse top-k query or completely avoiding it, if the m-NN products are found first.

6.1 Tighter Similarity Bounds

Given a point p that is going to be processed and a set of points (maintained with an in-memory R-tree I) with already computed reverse top-k sets, the question is how to tighten the bounds of p. Figure 4 illustrates an example that helps explaining this process. Let $\{p_1, \ldots, p_f\}$ denote the already computed points located in the two boxes defined by (a) the origin of the axes and p (*lower box*), and (b) p and the maximum corner of the data space (*upper box*). We claim that only a subset of these points need to be employed to update the bounds of p. From the upper box, we need only the points that belong to the skyline with respect to p. We denote this set of points $Domd(p)$, since they are dominated by p. In the Figure, $Domd(p) = \{p_6, p_7\}$. From the lower box, we need only the points that belong to the skyline with respect to p, if we consider p as the origin of the space. We denote this set of points $Dom(p)$, since they dominate p. In the figure, $Dom(p) = \{p_3, p_4, p_5\}$.

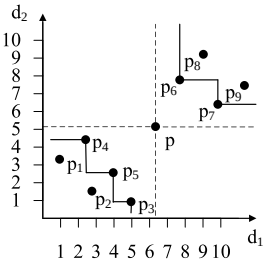


Fig. 4. Example of necessary points to compute tighter bounds.

We compute the intersection of reverse top-k results of points in $Dom(p)$, and the union of reverse top-k results of points in $Domd(p)$. Let $r_{Dom(p)}$ and $r_{Domd(p)}$ denote the corresponding binary vectors. Our claim is that these vectors can be utilized to compute tighter bounds for p, as explained by the following lemmas. (If one of these sets happens to be empty, then the corresponding bound is not updated.)

Lemma 7. *Given a point p and the set of points $p_i \in Dom(p)$, it holds that $RTOP_k(p) \subseteq \bigcap_{\forall p_i \in Dom(p)} RTOP_k(p_i)$*

Proof: It holds that $\forall p_i \in Dom(p)$: $p_i \prec p$. Moreover, from Lemma 2 we derive that: $\forall p_i : RTOP_k(p) \subseteq RTOP_k(p_i)$. According to set theory it holds that: if $A \subseteq B$ and $A \subseteq C$, then $A \subseteq B \cap C$. Consequently, it holds that $RTOP_k(p) \subseteq \bigcap_{\forall p_i \in Dom(p)} RTOP_k(p_i)$. □

Lemma 8. *Given a point p and the set of points $p_i \in Domd(p)$, it holds that $\bigcup_{\forall p_i \in Domd(p)} RTOP_k(p_i) \subseteq RTOP_k(p)$*

Proof: It holds that $\forall p_i \in Domd(p)$: $p \prec p_i$. Moreover, from Lemma 2 we derive that: $\forall p_i : RTOP_k(p_i) \subseteq RTOP_k(p)$. According to set theory it holds that: if $B \subseteq A$ and $C \subseteq A$, then $B \cup C \subseteq A$. Consequently, it holds that $\bigcup_{\forall p_i \in Domd(p)} RTOP_k(p_i) \subseteq RTOP_k(p)$. □

According to Lemmas 7 and 8, we are able to define a virtual MBR M^* that contains p, whose lower corner's vector is equal to $r_{Dom(p)}$ and higher corner's vector is equal to $r_{Domd(p)}$ vector. Then, we compute the $max\_sim(M^*, q)$ and $min\_sim(M^*, q)$ to query point q, as we have presented in section 4.2, replacing r_l with $r_{Dom(p)}$ and r_u with $r_{Domd(p)}$, respectively.

In the case of extended Jaccard, by utilizing the set $Dom(p)$, we compute the $r_{Dom(p)}$ assigning to its i-th coordinate the highest value (i.e., lowest ranking) that appears in the i-th position of the products' vectors that belong to the $Dom(p)$ set. For the construction of $r_{Domd(p)}$ vector we set the i-th coordinate equal to the lowest value (i.e., highest ranking) among the values of products' vectors contained in $Domd(p)$ set. Thus, we define a virtual MBR M^*that contains p, with its lower and higher corner's reverse top-k sets represented by the $r_{Dom(p)}$ and $r_{Domd(p)}$, respectively. Using the formulas for the similarity bounds presented in section 5 we compute the bounds, by again replacing r_l with $r_{Dom(p)}$ and r_u with $r_{Domd(p)}$.

6.2 Improved Query Processing

Based on the above, we improve the performance of our query processing algorithms by updating the bounds of a point, before we execute the reverse top-k query. In more detail, when our algorithms reach a leaf MBR, a set of reverse top-k queries needs to be processed (one for each enclosed point). Before we execute each query for point p, we first retrieve sets $Dom(p)$ and $Domd(p)$ by querying the in-memory R-tree I, compute vectors $r_{Dom(p)}$ and $r_{Domd(p)}$ respectively, and update the bounds of p as described above. Since the bounds become more tight, we can often avoid the execution of the reverse top-k query.

6.3 Handling Updates

Each time a new customer c is inserted, we need to perform a TOP_k query for c and update appropriately the r vectors of those MBR corners which are affected by the insertion of that customer. In more detail, there is no change for the r vectors of MBR corners that are dominated by the k_{th} product in the favorite list of customer c. Thus, we evaluate an $RTOP_k$ query for those corners of R-tree's MBRs which are not dominated by the k_{th} product in the TOP_k result set of customer c. On the other hand, when a customer is deleted, we only set to zero the entries of all the r vectors that refer to the specific customer.

When a new product p is added to the data set P, we increase by one all the non-zero values of the r vectors, which represent MBR corners that are dominated by p. For those r vectors, representing the $RTOP_k$ set of MBR corners that dominate p, there is no action. All the remaining r vectors should be recomputed performing an $RTOP_k$ query for the corner they represent. If a product p is deleted, we follow the same process as above. The only difference is that we decrease instead of increase the non-zero values of the r vectors, which represent MBR corners that are dominated by p. In order to minimize the cost by the $RTOP_k$ queries that need to be executed, we may perform a batch mode insertion/deletion of products. This would lead to only one $RTOP_k$ query execution for every MBR corner, no matter what the number of insertions/deletions in data set P is. In case of two or more R-tree nodes are merged or an R-tree node splits, we execute an $RTOP_k$ for all the corners of the new MBRs created after the merge/split process.

7 Experimental Evaluation

In this section, we first describe the results of two user study evaluations that we performed in order to assess the effectiveness of our proposed method. We then present an extensive experimental evaluation of the proposed user-centric similarity search algorithms on various datasets. All variants of the presented algorithms are implemented in Java and the experiments ran on a desktop PC with an i7 CPU (4 cores, 3.4 GHz), 8GB RAM, and 128GB SSD.

7.1 User Studies Evaluation

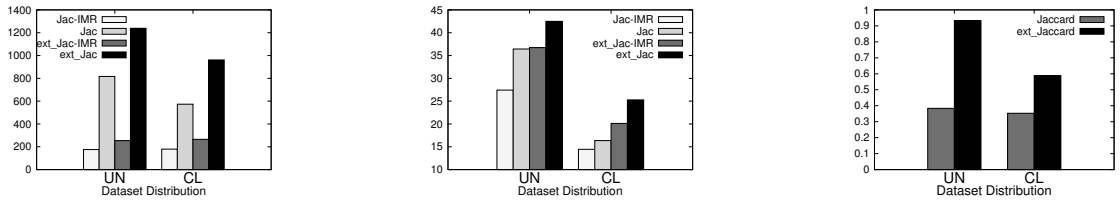
We first initiated a small user study, where 30 academic users (i.e. Ph.D. students and faculty) were asked to answer a questionnaire. We utilized a basketball dataset, which contains five statistics (points, rebounds, assists, steals, blocks) of the annual performance of NBA players. We selected five users and we asked each one of them to define a weighting vector that expresses their preferences about the performance of a center/forward NBA player. We then asked them to do the same for the case of players in a point/shooting guard position. We then selected randomly 20 queries/players (10 players from each group of center/forwards - point/shooting guards) and we executed our $1-NN$ algorithm using the extended Jaccard metric, in order to obtain the most similar player for every query. Then, all the 30 users were asked to choose between the $1-NN$ that our method returned and the $1-NN$ obtained according to the Euclidean distance similarity metric. In most cases (13 out of 20 queries) the majority of users chose the player that our method suggested. Only for five queries the majority of the users selected the player that the method using the Euclidean distance similarity metric returned. For the two remaining queries both methods' results are chosen by the same number of users.

In order to evaluate this use case scenario on a larger population, we utilized the well known crowd sourcing platform *CrowdFlower* and we launched a task asking from 220 contributors/users to answer each one of the aforementioned questions. A distinct difference from the previous study is that the group of 30 users that were initially used had been selected based on their familiarity with NBA. On the other hand, we had no control on the selection of contributors that performed the user study on the crowd sourcing platform. Still, the results were quite similar. Our method's suggestions were chosen by the majority of the users for 10 questions, while in one question the two alternatives were selected by the same number of users. Even though for the remaining 9 queries the users chose to select the answer suggested by the Euclidean distance, we observed that in many cases the differences in the user's opinions between that selection and our method's suggestion were small. On the contrary, for 6 out of 10 queries 70% or more of the the users preferred our method. For the Euclidean distance, there was only one query where more than 65% of the users had chosen that result. We believe that the results from the two user studies are encouraging and we leave a more thorough evaluation on this and other datasets as feature work.

7.2 Experimental Setup

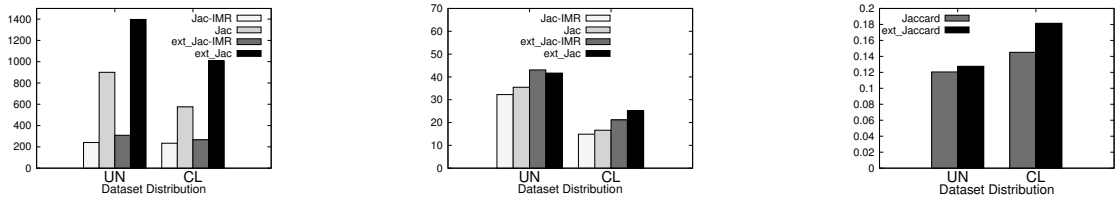
Data sets. In the experimental study, we employ both real and synthetic data sets. In the case of synthetic data sets, we generated products using uniform (UN) and clustered (CL) distributions. In either case, the generated values of each attribute belong to the $[0, 10K]$ range. For the UN data set, all attribute values are generated independently using a uniform distribution. In order to create the CL data set, we first create C_S cluster centroids and then we generate d values (assigned to each one of the data vector's coordinates) that follow a normal distribution with variance σ_S^2, and a mean equal to the corresponding coordinate of the centroid. The data set P is indexed using a disk-resident R-tree with block (node) size set to 8KB and a buffer of 100 nodes.

We also used four real data sets namely NBA, WINE, COLOR and HOUSE. NBA consists of 17265 5-dimensional tuples representing the statistics of NBA players for five different categories. WINE is a 10-dimensional data set from UCI Machine Learning Repository that contains the values of ten characteristics (i.e. alcohol, magnesium, color intensity, total phenols) of 6497 different types of wines. COLOR consists of 68040 9-dimensional tuples describing features of images in HSV color space. HOUSE is a 6-dimensional data set, that contains the percentage of 127930 American families' annual income spent on 6 types of expenditure (i.e. gas, electricity, water, heating).



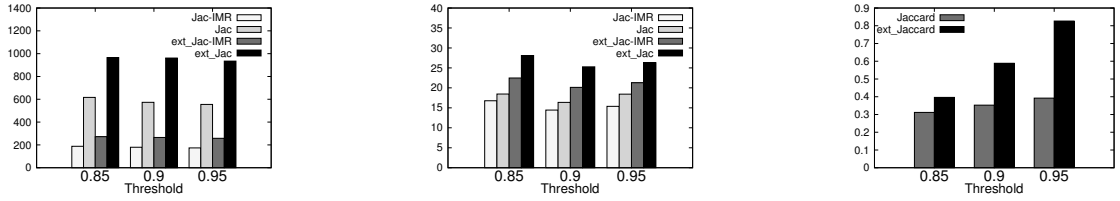
(a) $\#RTOP_k$ Evaluations (b) Execution Time (secs) (c) $sim_{UC,EU}$

Fig. 5. $\theta-$Similarity Queries Performance.



(a) $\#RTOP_k$ Evaluations (b) Execution Time (secs) (c) $sim_{UC,EU}$

Fig. 6. $m-NN$ Queries Performance



(a) $\#RTOP_k$ Evaluations (b) Execution Time (secs) (c) $sim_{UC,EU}$

Fig. 7. θ-Similarity Queries Performance Varying Threshold θ.

For the data set W of the weighting vectors, we used a clustered (CL) distribution. For its creation, first C_W cluster centroids that belong to the (d-1)-dimensional hyperplane, defined by $\sum w_i = 1$, are selected randomly. Then, each coordinate is generated on the (d-1)-dimensional hyperplane by following a normal distribution on each axis with variance σ_W^2, and a mean equal to the corresponding coordinate of the centroid. For both W and P datasets we used the clustered (CL) distribution with five clusters and variance 0.05^2 as the default setting.

Algorithms. We evaluate our user-centric similarity search algorithms (θ-similarity and m-NN) both for the Jaccard and the extended Jaccard similarity metric. We also implemented the variant that exploits already processed $RTOP_k$ queries (maintained in an in-memory R-tree) to improve the bounds, as described in Section 6. The contents of this in-memory R-tree are discarded after completion of the corresponding query. Experiments using the in-memory R-tree are depicted using the "-IMR" suffix in the labels of the corresponding graphs.

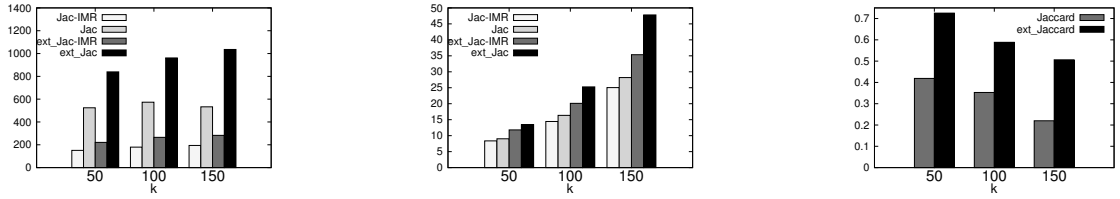
Metrics. Our main metrics include: a) the number of reverse top-k queries processed, b) the query execution time required by each algorithm, and c) the similarity $sim_{UC,EU}$ of the result to the answer that would have been obtained by using the Euclidean distance. This similarity is expressed as the Jaccard coefficient of the result sets (number of common products in the answers over the size of their union) of the corresponding queries. For the m-NN queries in Euclidean space we used the algorithm of [7]. For the range queries we executed a Euclidean NN query for the same product q the θ-similarity query examines and retrieved as many products as the size of the $\theta-$similarity query's result set. In our experimental evaluation, a value of $sim_{UC,EU}$ that is much lower than one indicates that the user-centric similarity query (m-NN/θ-similarity) returns significantly different answers.

Queries. We present average values over 20 queries in all cases. The query points are randomly selected from a subset of the data points in P. To increase the probability of having non-empty result sets, this subset contains either the closest to axes origins products of P in case of synthetic data set, or products from the skyline set of P for the real data sets.

Parameters. We conduct experiments varying the dimensionality d (2-6), the cardinality $|P|$ (10K-100K), the cardinality $|W|$ (1K-15K), the value of k (50-150), the value of θ (0.7-0.95) and the data distributions for P. We set the default values for the experimental parameters as follows: $k = 100$, $|W| = 1000$, $\theta = 0.9$, $d = 4$ and $m = 10$.

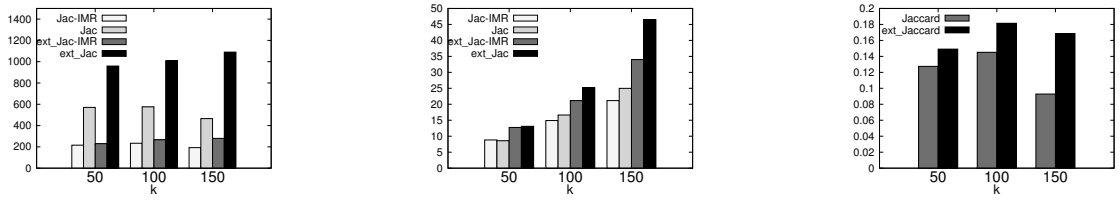
7.3 Experimental Results

Uniform vs Clustered set P. In Figure 5 we study the performance of the $\theta-$similarity queries for the two different data sets. The variant of the proposed algorithm for $\theta-$similarity queries, which utilizes the in-memory R-tree, reduces the $RTOP_k$ evaluations significantly (at least three times less), as shown in Figure 5a.



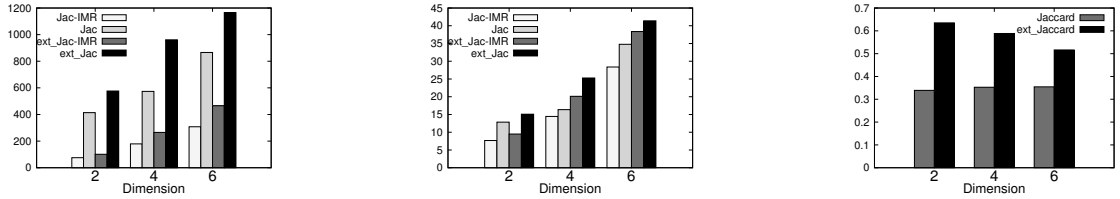
(a) $\#RTOP_k$ Evaluations (b) Execution Time (secs) (c) $sim_{UC,EU}$

Fig. 8. θ-Similarity Queries Performance Varying k.



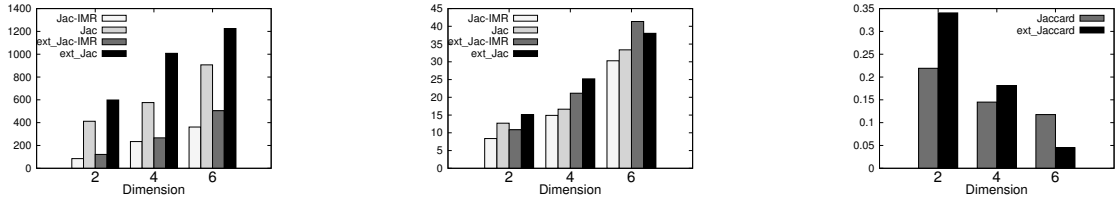
(a) $\#RTOP_k$ Evaluations (b) Execution Time (secs) (c) $sim_{UC,EU}$

Fig. 9. m-NN Queries Performance Varying k.



(a) $\#RTOP_k$ Evaluations (b) Execution Time (secs) (c) $sim_{UC,EU}$

Fig. 10. θ-Similarity Queries Performance Varying d.



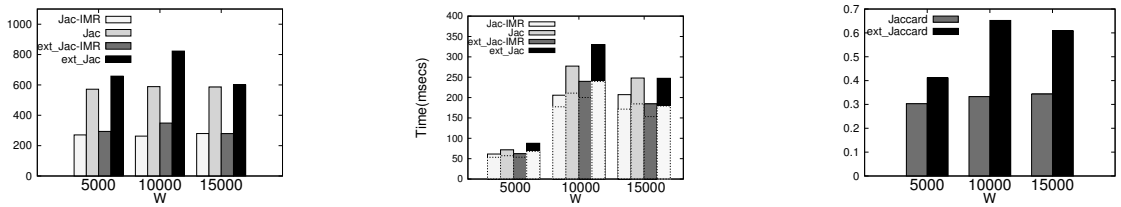
(a) $\#RTOP_k$ Evaluations (b) Execution Time (secs) (c) $sim_{UC,EU}$

Fig. 11. m-NN Queries Performance Varying d.

In consequence, this reduces execution time (Figure 5b). Figure 5c serves as a comparison of user-centric similarity search with traditional similarity search using the Euclidean distance. The chart clearly shows that the retrieved results are different. Figure 6 repeats the experiment, using this time m-NN queries, for both cases of data distribution. The same trends, as in the case of $\theta-$similarity queries, between the original algorithm and the one that uses the in-memory R-tree are observed.

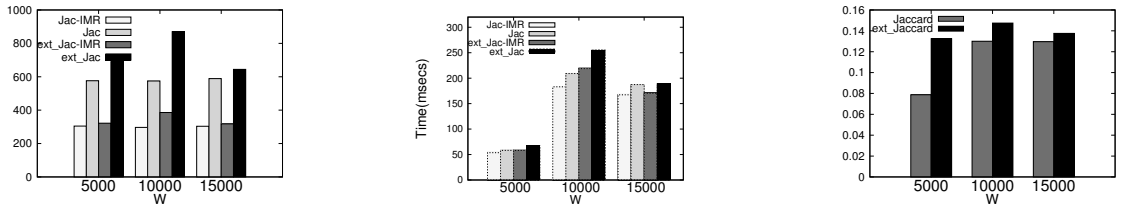
Varying θ. In Figure 7 we study the sensitivity of $\theta-$similarity queries, when varying the threshold value θ. Higher similarity threshold values result in a slight decrease in the number of $RTOP_k$ executions (Figure 7a) as well as the execution time (Figure 7b). When we consider the similarity between the results sets of proposed queries and similarity queries in Euclidean space, in Figure 7c, we observe that higher threshold values increase the similarity of the two result sets.

Varying k. The effect of the increase in values of k to our algorithms is shown in Figures 8, 9. A key observation is that the execution time (Figures 8b, 9b) increases with k as an effect of the cost implied by the reverse top-k queries execution. The reason is that each reverse top-k query has a larger result set and becomes more expensive, since users have more products in their top-k result sets. The number of $RTOP_k$ (Figures 8a, 9a) is also increasing for higher values of k. When metric $sim_{UC,EU}$ is considered (Figures 8c, 9c), we observe that the higher the values of k the lower the similarity between the results of user centric queries and similarity queries in Euclidean space. In case of m-NN queries we cannot observe the same trend because of the



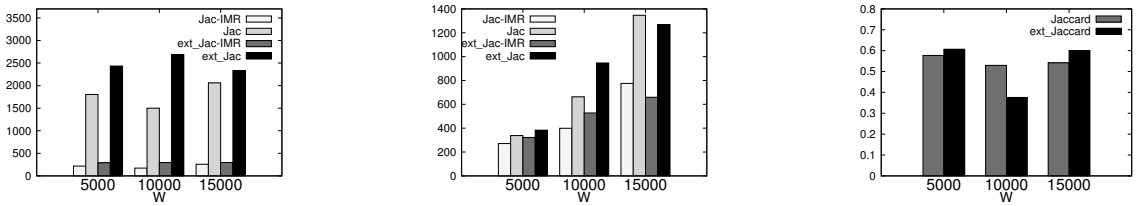
(a) $\#RTOP_k$ Evaluations (b) Execution Time (secs) (c) $sim_{UC,EU}$

Fig. 12. θ-Similarity Queries Performance Varying W, $|P|$=10K.



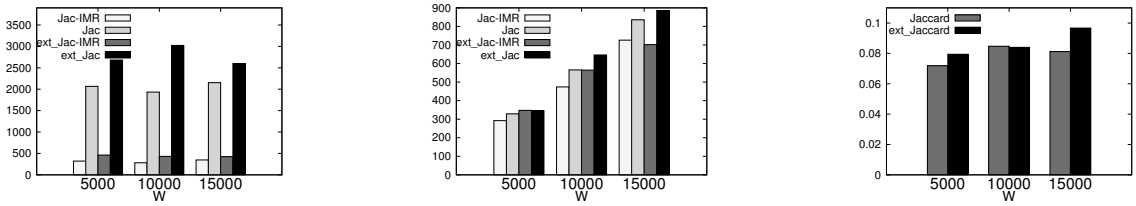
(a) $\#RTOP_k$ Evaluations (b) Execution Time (secs) (c) $sim_{UC,EU}$

Fig. 13. m-NN Queries Performance Varying W, $|P|$=10K.



(a) $\#RTOP_k$ Evaluations (b) Execution Time (secs) (c) $sim_{UC,EU}$

Fig. 14. θ-Similarity Queries Performance Varying W, $|P|$=100K.



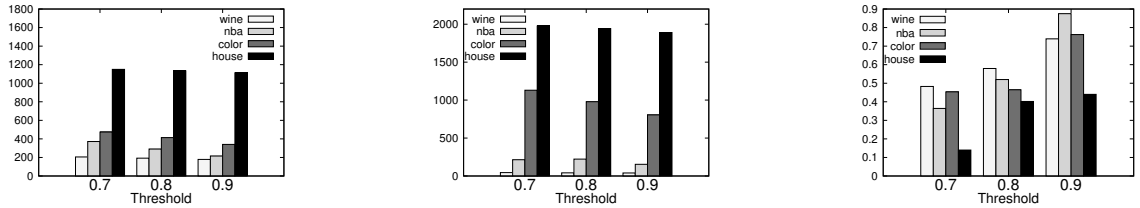
(a) $\#RTOP_k$ Evaluations (b) Execution Time (secs) (c) $sim_{UC,EU}$

Fig. 15. m-NN Queries Performance Varying W, $|P|$=100K.

extremely low similarity of the aforementioned result sets.

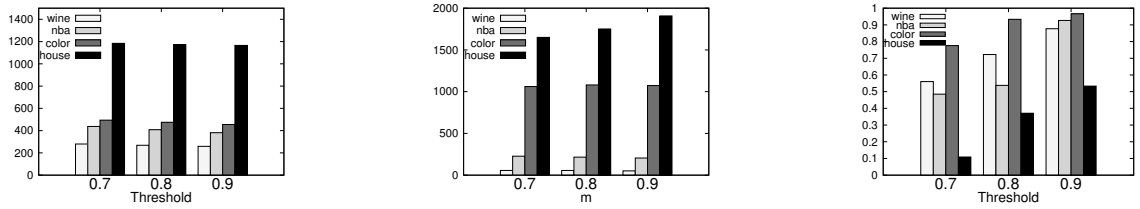
Varying dimensionality d**.** Figures 10, 11 present the behavior of the algorithms, while the dimensionality of data sets increases. Increasing dimensionality generally results in increased number of $RTOP_k$ executions (Figures 10a, 11a) as well as increased execution times (Figures 10b, 11b). Results also tend to be more dissimilar compared to those obtained using the Euclidean distance (Figures 10c, 11c), as a result of the increased sparsity that tends to put products further apart in higher dimensions.

Varying $|W|$**.** Figures 12-15 depict the behavior of our algorithms for increasing values of $|W|$ in case two different instances for data set P are utilized (the first one has the default size, while the second is much larger with $|P| = 100K$). In both scenarios, when the size of W increases, the execution time also increases (Figures 12b, 13b, 14b, 15b) because of the higher execution cost of each $RTOP_k$ query imposed by the increasing number of weighting vectors that needs to be examined to obtain its result set. Both, the number of $RTOP_k$ executions and $sim_{UC,EU}$ are not especially affected by the increase of $|W|$ presenting a relatively stable behavior as it is depicted in Figures 12a, 13a, 14a, 15a and Figures 12c, 13c, 14c, 15c, respectively. Moreover, we show that the execution time for both of the algorithms is dominated by the time cost of $RTOP_k$ queries execution. In Figures 12b, 13b inner boxes with the dashed-lined borders present the time consumed during our algorithms execution for the evaluation of the $RTOPk$ queries. Regarding the time cost for the computation of both



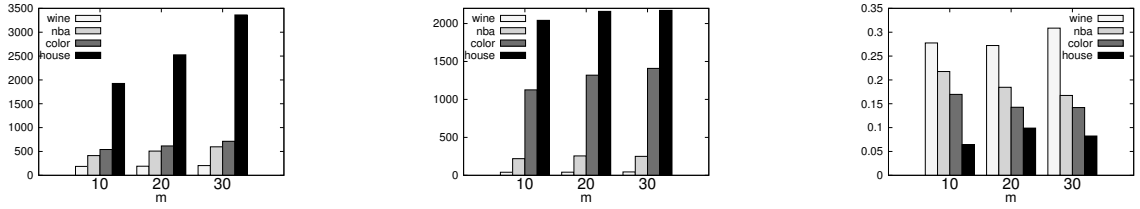
(a) $\#RTOP_k$ Evaluations (b) Execution Time (secs) (c) $sim_{UC,EU}$

Fig. 16. θ-Similarity Queries Performance using Jac-IMR.



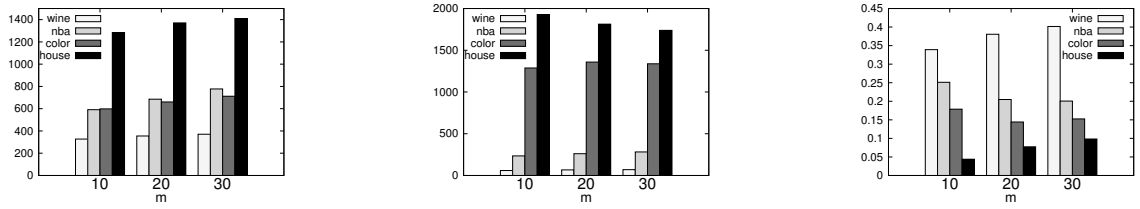
(a) $\#RTOP_k$ Evaluations (b) Execution Time (secs) (c) $sim_{UC,EU}$

Fig. 17. θ-Similarity Queries Performance using ext_Jac-IMR.



(a) $\#RTOP_k$ Evaluations (b) Execution Time (secs) (c) $sim_{UC,EU}$

Fig. 18. m-NN Queries Performance using Jac-IMR.

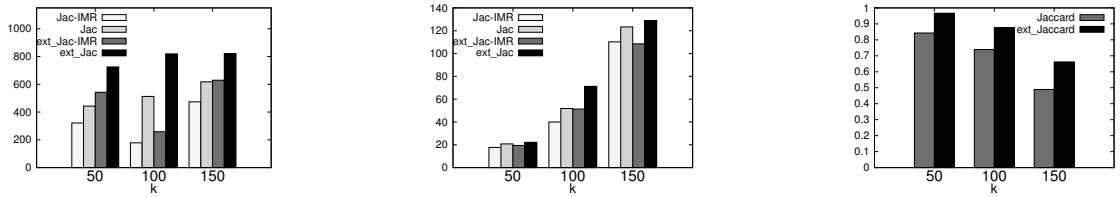


(a) $\#RTOP_k$ Evaluations (b) Execution Time (secs) (c) $sim_{UC,EU}$

Fig. 19. m-NN Queries Performance using ext_Jac-IMR.

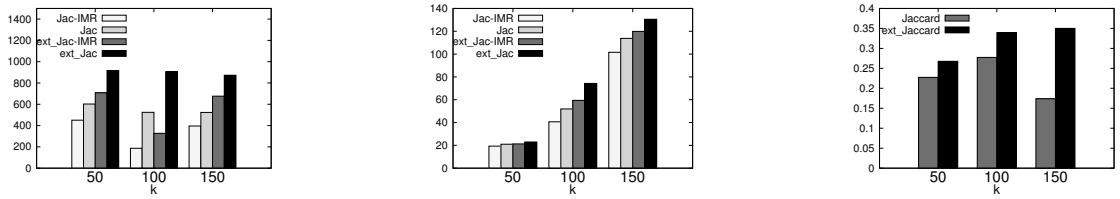
skyline and dominated sets utilized by algorithms' IMR variants are so small (i.e. varies from 300 to 500 msecs) that is not visible in the plots.

Real Data Sets. In Figures 16-19, we present results for the in-memory R-tree variants of our methods, using the real data sets and varying the values of threshold and m for the $\theta-$similarity and m-NN queries, respectively. Results from all the four real data sets confirm the study with the synthetic data sets, as they follow the same trends with the number of $RTOP_k$ evaluations and execution time decrease when the threshold increases. In Figures 16c, 17c, 18c and 19c, we can observe the low values of $sim_{UC,EU}$ for all the real data sets, especially the extremely low values for the HOUSE dataset in case of the m-NN queries (up to 7%), showing clearly that different results can be obtained when exploiting the user preferences for similarity search. Moreover, in Figures 20, 21 we present the performance for all the variants of our methods utilizing the WINE dataset while we vary the value of k. As in the case of synthetic datasets, we observe that when we increase k the query execution time for both of our algorithms increases (figures 20b, 21b) while the $sim_{UC,EU}$ decreases in $\theta-$similarity query (Figure 20c) and remains relatively stable (Figure 21c) for the m-NN query. In Figures 22, 23, when we vary value of $|W|$ for the WINE dataset, our methods performance follow the same trend as in sensitivity analysis study we performed using synthetic datasets. Thus, in Figures 22c, 23c is shown that $sim_{UC,EU}$ is not especially affected for both query types



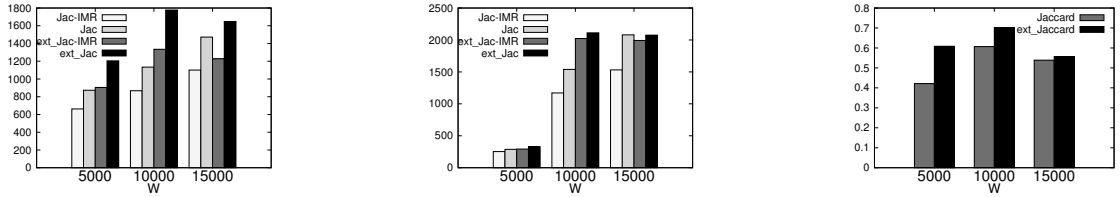
(a) $\#RTOP_k$ Evaluations (b) Execution Time (secs) (c) $sim_{UC,EU}$

Fig. 20. θ-Similarity Queries Performance Varying k (WINE Dataset).



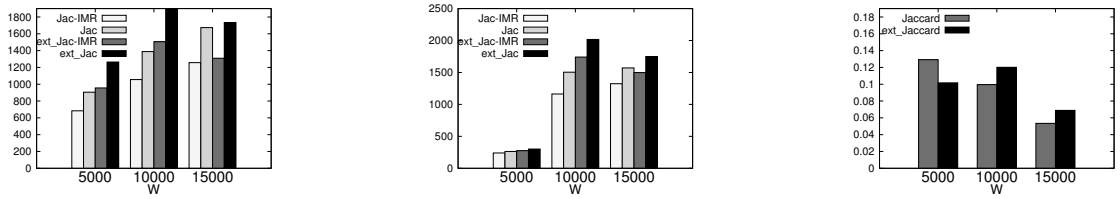
(a) $\#RTOP_k$ Evaluations (b) Execution Time (secs) (c) $sim_{UC,EU}$

Fig. 21. m-NN Queries Performance Varying k (WINE Dataset).



(a) $\#RTOP_k$ Evaluations (b) Execution Time (secs) (c) $sim_{UC,EU}$

Fig. 22. θ-Similarity Queries Performance Varying W (WINE Dataset).



(a) $\#RTOP_k$ Evaluations (b) Execution Time (secs) (c) $sim_{UC,EU}$

Fig. 23. m-NN Queries Performance Varying W (WINE Dataset).

while execution time (Figures 22b, 23b) and number of $RTOP_k$ (Figures 22a, 23a) increase for higher values of $|W|$.

8 Related Work

Top-k queries are a long studied topic in the database and information retrieval communities. Such queries return the k most promising products, based on available user preferences [6], [8]. The work of [9] addresses the problem of measuring the quality of top-k result sets returned by an information retrieval system, as is the case of comparing search engine results. The authors discuss several alternative measures and provide fast approximation algorithms for the evaluation of some of them. On the other hand, reverse top-k queries, introduced in [4], return the users that place a product (that is the query point) in their top-k result sets. A use for reverse top-k queries is to identify influential products, where influence is defined as the cardinality of the reverse top-k result set [10]. This definition of influence is useful for market analysis, since it is directly related to the number of customers that value a particular product. Despite recent techniques for evaluating reverse top-k queries, they are known to incur significant processing and I/O overhead, as such a query typically requires the execution of multiple top-k queries for computing the customers that prefer the queried product.

Another important class of queries are the Reverse Nearest Neighbor (RNN) queries [11], that are complement to the well known Nearest Neighbor queries [7]. The RNN queries are of particular interest in a wide range of applications such as decision support systems, profile based marketing and document databases.

A RNN query returns all objects whose k nearest neighbors contain the query object. The classical motivating example [12] of RNN is the decision support task of identifying the optimal location for a new store. Given several location choices, the strategy is to pick the location that can attract the maximum number of customers. A RNN query returns the customers who are likely to choose the new store because of its geographical proximity over the existing stores. There are several approaches for the implementation of RNN queries [13], [14]. There are also algorithms that work for higher dimensionality data sets [15]. What we suggest in our work is quite different from the similarity introduced and utilized in RNN queries, where the results are based solely on objects' characteristics while users' preferences are not taken into account.

Item-based collaborative filtering techniques [16] may share a similar intuition, but in contrary to our methods, they suggest that customers have a *taste* of some products and thus rate them. Suppose, the case of a product that recently launched in the market or a product is under designing during its manufacturing process. In both cases, there would be no ratings expressing customers' opinions, making a collaborative filtering algorithm inapplicable. On the other hand, our framework does not require any previous knowledge about users' opinions for the products because they express in a more general way their preferences providing a weighting factor for each attribute of products, which is different than rating individual products.

9 Conclusion

In this paper we introduced a user-centric similarity framework in which the similarity of products is assessed by taking into account user preferences. We demonstrated via examples and through our experiments that user-centric similarity search can yield quite different results than using conventional metrics that only look at the products, in isolation to the preferences their customers have expressed. We identified two interesting query types and we proposed efficient algorithms for their execution. We also discussed optimizations that help reduce execution times.

References

[1] A. Rajaraman and J. D. Ullman, *Mining of massive datasets*. Cambridge: Cambridge Univ. Press, 2012.

[2] K. Georgoulas and Y. Kotidis, "Towards Enabling Outlier Detection in Large, High Dimensional Data Warehouses," in *Proceedings of SSDBM*, 2012, pp. 591–594.

[3] H. Becker, M. Naaman, and L. Gravano, "Learning similarity metrics for event identification in social media," in *Proceedings of WSDM*, 2010.

[4] A. Vlachou, C. Doulkeridis, Y. Kotidis, and K. Nørvåg, "Reverse Top-k Queries," in *Proceedings of ICDE*, 2010.

[5] A. Guttman, "R-Trees: A Dynamic Index Structure for Spatial Searching," in *Proceedings of SIGMOD*, 1984, pp. 47–57.

[6] V. Hristidis, N. Koudas, and Y. Papakonstantinou, "PREFER: A System for the Efficient Execution of Multi-parametric Ranked Queries," in *Proceedings of SIGMOD*, 2001, pp. 259–270.

[7] N. Roussopoulos, S. Kelley, and F. Vincent, "Nearest Neighbor queries," in *Proceedings of SIGMOD*, 1995, pp. 71–79.

[8] R. Fagin, "Combining Fuzzy Information: an Overview," *SIGMOD Record*, vol. 31, no. 2, pp. 109–118, 2002.

[9] R. Fagin, R. Kumar, and D. Sivakumar, "Comparing Top-k Lists," *SIAM J. Discrete Math.*, vol. 17, no. 1, pp. 134–160, 2003.

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

[11] F. Korn and S. Muthukrishnan, "Influence Sets Based on Reverse Nearest Neighbor Queries," in *Proceedings of SIGMOD*, 2000, pp. 201–212.

[12] E. Achtert, C. Böhm, P. Kröger, P. Kunath, A. Pryakhin, and M. Renz, "Efficient Reverse k-Nearest Neighbor Search in Arbitrary Metric Spaces," in *Proceedings of SIGMOD*, 2006, pp. 515–526.

[13] F. Korn, S. Muthukrishnan, and D. Srivastava, "Reverse Nearest Neighbor Aggregates Over Data Streams," in *Proceedings of VLDB*, 2002, pp. 814–825.

[14] K. C. K. Lee, B. Zheng, and W.-C. Lee, "Ranked Reverse Nearest Neighbor Search," *IEEE Trans. Knowl. Data Eng.*, vol. 20, no. 7, pp. 894–910, 2008.

[15] A. Singh, H. Ferhatosmanoglu, and A. S. Tosun, "High Dimensional Reverse Nearest Neighbor Queries," in *Proceedings of CIKM*, 2003, pp. 91–98.

[16] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. Riedl, "Item-based collaborative filtering recommendation algorithms," in *Proceedings of WWW*, 2001.

Konstantinos Georgoulas received the BSc degree from the Department of Informatics of the Athens University of Economics and Business in 2002. In 2008 received the MSc from the Department of Informatics of the University of Ioannina. He is currently a PhD Student at Athens University of Economics and Business. His research interests include query processing and data management in large-scale systems.

Akrivi Vlachou received the BSc and MSc degrees from the Department of Computer Science and Telecommunications of University of Athens in 2001 and 2003, respectively, and the PhD degree in 2008 from the Athens University of Economics and Business (AUEB). She is currently a post-doctoral researcher at the University of Piraeus. Her research interests include query processing and data management in large-scale distributed systems.

Christos Doulkeridis received the BSc degree in electrical engineering and computer science from the National Technical University of Athens and the MSc and PhD degrees in Information Systems from the Department of Informatics of Athens University of Economics and Business. He is currently an assistant professor in the Department of Digital Systems of the University of Piraeus. His research interests include parallel and distributed data management, and data analytics.

Yannis Kotidis received the BSc degree in electrical engineering and computer science from the National Technical University of Athens, the MSc and PhD degrees in computer science from the University of Maryland. He is an associate professor in the Department of Informatics of Athens University of Economics and Business. Between 2000 and 2006, he was a senior technical specialist at the Database Research Department of AT&T LabsResearch in Florham Park, New Jersey. His main research interests include large scale data management systems, data warehousing and sensor networks.