Metric-Based Top-k Dominating Queries

Eleftherios Tiakas
Dept. of Informatics
Aristotle University of Thessaloniki
Thessaloniki, Greece
tiakas@csd.auth.gr

George Valkanas
Dept. of Informatics and Telecommunications
University of Athens
Athens, Greece
gvalk@di.uoa.gr

Apostolos N. Papadopoulos
Dept. of Informatics
Aristotle University of Thessaloniki
Thessaloniki, Greece
papadopo@csd.auth.gr

Yannis Manolopoulos
Dept. of Informatics
Aristotle University of Thessaloniki
Thessaloniki, Greece
manolopo@csd.auth.gr

Dimitrios Gunopulos
Dept. of Informatics and Telecommunications
University of Athens
Athens, Greece
dg@di.uoa.gr

ABSTRACT

Top-k dominating queries combine the natural idea of selecting the k best items with a comprehensive "goodness" criterion based on dominance. A point p_1 dominates p_2 if p_1 is as good as p_2 in all attributes and is strictly better in at least one. Existing works address the problem in settings where data objects are multidimensional points. However, there are domains where we only have access to the distance between two objects. In cases like these, attributes reflect distances from a set of input objects and are dynamically generated as the input objects change. Consequently, prior works from the literature can not be applied, despite the fact that the dominance relation is still meaningful and valid. For this reason, in this work, we present the first study for processing *top-k dominating queries* over distance-based dynamic asttribute vectors, defined over a *metric space*. We propose four progressive algorithms that utilize the properties of the underlying metric space to efficiently solve the problem, and present an extensive, comparative evaluation on both synthetic and real world data sets.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems

Keywords

Dominating queries, Metric space, Progressive algorithms.

1. INTRODUCTION

Preference-based queries [14] allow users to enforce additional constraints and better guide the object selection process. One way to express such preferences is to provide a scoring function over the object's attributes. Another way is to give some hints regarding the maximization or minimization of attribute values, without giving an explicit scoring function. Based on these alternatives, there are two classic preference-based query types that have been studied extensively in the literature: (i) *top-k* and (ii) *skyline queries*.

Top-k query processing has been an active research area spanning disciplines like Web search, p2p-based retrieval, multimedia databases, to name a few. The query's power lies in its flexibility, supporting user-defined and ad-hoc scoring functions, and its ability to bound the number of results through the parameter k. Unfortunately, the selection of a meaningful scoring function is not always easy, since different scoring functions generally produce different results. Moreover, top-k queries are sensitive to value scaling.

On the other hand, skyline queries rely on the *dominance* property. More specifically, p dominates q, if p is at least as good as q in every attribute and it is strictly better that q in at least one of them. The most important advantage of skyline queries is that no magic parameters or scoring functions are required. The result also remains unaffected by attribute scaling. However, the result is a set, i.e., no inherent ranking of the points is supported, and its size depends on the data set's underlying properties.

It is easy to realize that these preference-based queries are complementary. In an attempt to combine their advantages and cancel out their disadvantages, a hybrid approach was proposed in [24, 25]: the *top-k dominating query*. This new query ranks objects (as in top-k queries) according to a scoring function that relies on dominance (as in skyline queries): The score of an object p_i equals the number of points that p_i dominates. Overall, *top-k dominating queries* exhibit the following desirable properties: *i*) the result size is controllable, *ii*) it is scale invariant, *iii*) it is based on an intuitive score value for each object, and *iv*) no other user-defined scoring function is required.

The importance of this query has been prominently exemplified by the numerous works on the topic, and is partly due to its simple, yet intuitive explanation: the more points that a point p dominates, the better p is. Moreover, it simulates how users select items, in lack of better alternatives: e.g., if the "best" camera is out-of-stock, the second best is picked, etc. The query was initially proposed in [24] and

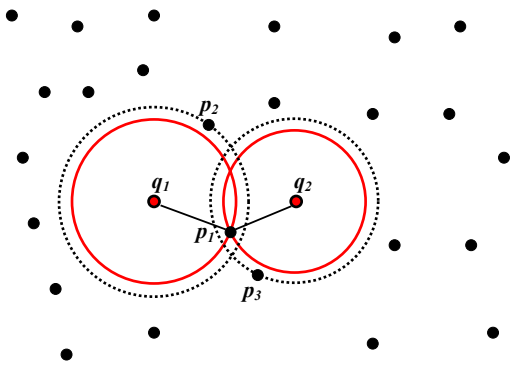


Figure 1: A dominating query with two query objects q_1 and q_2.

later improved in [25]. It has been found useful in uncertain databases [16] and in streaming settings [15]. Its practicality was also demonstrated in subspace dominance queries [22].

Unfortunately, all these techniques work under the assumption that each object is associated with an attribute vector, and that dominance is based on the attribute values. This requirement renders them inapplicable in domains where vector-based representation is impossible or inefficient, e.g., biology, graphs, etc. For instance, consider a graph that captures object interactions e.g., a protein-protein interaction network; DNA sequences are another case, commonly represented by aminoacid strings.

For this reason, we study, for the first time, algorithms for top-k dominating queries where objects are defined in any *metric space* and the attribute values are generated *dynamically* based on a distance measure respecting the *metric space properties*. The dynamic nature of attributes poses significant challenges in comparison to the static case, because attribute values are not known in advance and, therefore, traditional multidimensional indexing schemes (e.g., R-trees) cannot be employed to provide efficient access.

Essentially, a top-k dominating query in a metric space accepts as input a set of user-selected objects $\{q_1, q_2, ..., q_m\}$ and returns a set A with k objects, that have the highest dominance scores according to the distance measure used. For example, in a protein-protein interaction network, the query points could be proteins or effector molecules that enhance / inhibit a protein's activity. The distance between objects could be defined by shortest paths, maximum flow or other measures [17], and could represent how well proteins interact with the query objects. Proteins that interact more with all query points are better than points which interact less. A top-3 dominating query will return the 3 proteins which are more frequently better at interacting with the query points. The result could indicate some underlying structural or functional commonalities, which other researchers can exploit.

To illustrate our idea further, Figure 1 shows a top-3 dominating query in a metric space with 25 2-dimensional points, under the Euclidean distance. Two query objects q_1 and q_2 are also depicted. Point p_1 is definitely the top-1 object, as it is the nearest neighbor of both query points q_1, q_2. Since no other point lies inside either $C_1(q_1, d(q_1, p_1))$ or $C_2(q_2, d(q_2, p_1))$, p_1 dominates all other points and its dominance score is $dom(p_1) = 24$. The second nearest neighbor of q_1 is p_2, whereas that of q_2 is p_3. Since $d(p_2, q_1) < d(p_3, q_1)$ (p_3 lies outside $C_3 q_1, d(p_2, q_1))$) and $d(p_2, q_2) > d(p_3, q_2)$ (p_2 lies outside $C_4(q_2, d(p_3, q_2))$), p_2 and p_3 do not dominate each other. However, p_2 and p_3 dominate all other points since there are no points lying inside the corresponding dotted circles (except p_1). Thus their dominance score is $dom(p_2)$=22 and $dom(p_3)$=22, whereas remaining points have a dominance score less than 22. Therefore, the set $\{p_1, p_2, p_3\}$ is the final answer to the top-3 dominating query based on query points q_1 and q_2 .

Finally, a desirable property in preference-based query processing is *progressiveness*. A progressive algorithm determines the best object first, then the second best and so on, until all k objects composing the answer are determined. Moreover, the user may stop the query execution if she thinks that the number of returned results is adequate. Our techniques enable progressive computation, providing non-blocking query execution plans.

Contributions. In summary, the contributions of our work are summarized as follows:

1. We introduce, for the first time, the concept of (dynamic) top-k dominating queries in metric spaces, generalizing the concept of dominating queries in multidimensional data sets.

2. We propose efficient techniques for answering metric-based top-k dominating queries, all of which satisfy the *progressiveness* requirement.

3. We present a detailed performance evaluation of our techniques, based on real-life and synthetic data sets, under different distance functions. Our pruning-based algorithms (`PBA1` and `PBA2`) offer a performance improvement by one to three orders of magnitude, compared to the alternatives, provided that the underlying metric access method supports incremental nearest-neighbor to ensure progressiveness.

Roadmap. The rest of the article is organized as follows: Section 2 describes briefly related work in the area. Section 3 presents some preliminary concepts regarding the topic of research and Section 4 studies in detail the query processing algorithms `SBA`, `ABA`, `PBA1` and `PBA2` as well as the theoretical infrastructure that these algorithms are based on. Section 5 offers performance evaluation results based on diverse data sets and distance functions. Finally, Section 6 concludes the work and discusses briefly future work in the area.

2. RELATED WORK

Top-k dominating queries have been proposed in [24, 25] as an alternative to general top-k and skyline queries. The ranking provided by this query is quite intuitive, it does not require specialized scoring functions and the size of the output is controlled by the parameter k. Among the algorithms studied, CBT shows the best overall performance. However, it requires the existence of an aggregate R-tree index and lacks progressiveness, since all k points must be first determined before the answer is returned. In addition, query processing involves all the available dimensions, which is quite restrictive taking into account that users usually focus on few attributes.

In [16] an algorithm has been proposed supporting top-k dominating queries in uncertain databases. The proposed approach has the same limitations with the previous one, since it is again based on aggregate R-trees. The novel features of this method is that it handles uncertainty in a clear and meaningful way in applications where uncertainty cannot be avoided. In [26] a randomized algorithm is proposed supporting probabilistic top-k dominating queries in uncertain data. Again, the proposed approach is based on aggregate R-trees. The proposed algorithm is highly accurate when the data cardinality and the dimensionality is low. The concepts of top-k dominating queries have been also used in [21]. That work studies web service discovery issues by using dominance relationships through multi-criteria matching. Finally, continuous monitoring of top-k dominating query results has been studied in [15] by taking a sliding window approach.

The common characteristic of the aforementioned approaches is that they are based on vector spaces, where the concept of dominance is directly applied on attribute values, which are a priori available. There is no work in the literature studying the problem of dominating query processing under the dynamic scenario where attribute values are generated on-the-fly representing distances from user-defined query objects. Although skyline queries have been studied over dynamic attributes [20] and metric spaces [6, 7, 12, 9], there is no work studying the problem of top-k dominating query processing in a metric space, where coordinates are dynamically defined by means of a metric function. This seems a natural generalization, taking into account that many modern applications rely solely on triangular inequality to support similarity queries [5].

In addition to the absence of algorithms for metric-based dominating queries, another limitation of previously proposed methods is that they lack *progressiveness*. Progressiveness is an important property since it enables the incremental production of results (relevant objects are retrieved one-by-one as soon as they are available). The only related progressive algorithm is Branch-and-Bound Skyline (BBS) [18] which has been designed to answer skyline queries in multi-dimensional data sets indexed by an R-tree. Thus, BBS is not applicable in our case. Moreover, the algorithm in [7] although works with metric-based access methods it is designed to report the skyline, and it is not equipped to handle the concept of dominance score.

In [22] the authors study efficient progressive algorithms for top-k dominating queries in multi-dimensional data sets, where the user may select a subset of the available dimensions. These techniques assume a vertical decomposition of the data set, where each dimension is organized separately, and therefore, they are not applicable in the metric case explored in this paper. However, some results from [22] are adapted and utilized in the present work, in order to provide: (i) efficient score computation and (ii) effective pruning.

3. BASIC CONCEPTS

In this section, we present some basic concepts and definitions regarding the focus of our research, in order to keep the work self-contained. Table 1 depicts the basic notations that are frequently used in the sequel. In general, sets of objects and query results are depicted by uppercase letters.

Let D be a set of data objects and $d()$ a function such that: $d : D \times D \longrightarrow \mathbb{R}$, which quantifies the *dissimilarity* between data objects satisfying the following properties:

Symbol	Description
D	the set of data objects
$n = \|D\|$	the number of data objects
Q	the set of query objects
$m = \|Q\|$	the number of query object
$d()$	a metric distance function
$NN(q,k)$	k-NN of object q
$ANN(Q,k)$	top-k aggregate NN objects of Q
$adist(p,Q)$	aggregate distance of object p from Q
$p \prec r$	object p dominates object r
$dom(p)$	domination score of p respecting Q
$MSS(Q)$	metric space skyline with respect to Q
$MSD(Q,k)$	top-k dominating objects with respect to Q

Table 1: Frequently used symbols.

$\forall p, q \in D, d(p,q) \geq 0$ (*positivity*)
$\forall p, q \in D, d(p,q) = d(q,p)$ (*symmetry*)
$\forall p \in D, d(p,p) = 0$ (*reflexivity*)
$\forall p, q, x \in D, d(p,q) \leq d(p,x) + d(x,q)$ (*triangular inequality*)

Then, d is called a *metric function* and the pair (D,d) is called a *metric space*. In case that the objects of the metric space are tuples (records) with numeric attributes, then the pair (D,d) is called a *vector space*, and, among others, any L_p norm may be used as the distance function. Note that a vector space is a special case of a metric space.

DEFINITION 1. *Let D be a data set, $q \in D$ and $d()$ be a distance function. A k-nearest neighbor query based on q, denoted as $NN(q,k)$, determines the k closest objects with respect to q. Formally: $NN(q,k) = A : |A| = k \wedge \forall p \in A, \forall x \in (D - A)$, $d(q,p) \leq d(q,x)$.*

DEFINITION 2. *If (D,d) is a metric space, Q is a set of query objects $Q = \{q_1, q_2, ..., q_m\}$ and $f()$ is a monotonically increasing function then the aggregate distance between an object p and the query set Q is defined as follows:*

$$adist(p,Q) = f(d(p,q_1), d(p,q_2), ..., d(p,q_m))$$

The result of an *aggregate nearest neighbor query*, denoted as $ANN(Q,k)$, contains the k objects with the minimum aggregate distance computed based on the distances from Q. An important factor is the selection of the *aggregate function* $f()$, which affects the performance of these queries. Commonly used aggregate functions are min, max, avg and sum (see [19]).

DEFINITION 3. *Let (D,d) be a metric space and Q be a set of query objects $Q = \{q_1, q_2, ..., q_m\}$. For any two objects $p, r \in D$, p dominates r ($p \prec r$) if the following condition holds:*

$$\forall i : 1 \leq i \leq m, d(p,q_i) \leq d(r,q_i) \ \wedge \ \exists q_j \in Q, d(p,q_j) < d(r,q_j)$$

Therefore, p dominates r, if and only if p has an equal or smaller distance than r to all query objects $q_i \in Q$, and p has a smaller distance than r to at least one query object. The set of objects in D which are *not dominated* by any other object (according to the distances from Q) is called the *metric space skyline* with respect to Q, denoted as $MSS(Q)$.

In [24] top-k dominating queries were defined for objects with fixed coordinates. In this paper, we generalize this concept by considering metric space dominating queries, denoted as $MSD(Q, k)$. We use the function $dom(p)$ to denote the number of objects dominated by an object p with respect to a particular query set, i.e., $dom(p) = |\{r \in D : p \prec r\}|$. The answer to such a query consists of the set of k objects that have the highest scores with respect to the ranking function $dom()$. In some cases, the distance vectors of two or more objects with respect to a query set may be the same. In this case, we say that these objects are *equivalent*:

DEFINITION 4. *Two objects p_1 and p_2 are called equivalent with respect to a query set $Q = \{q_1, q_2, ..., q_m\}$, if $\forall i : 1 \leq i \leq m$, $dist(p_1, q_i) = dist(p_2, q_i)$.*

In the following section, we study progressive algorithms for efficient processing of metric space dominating queries. The performance of the algorithms is measured by considering the CPU and I/O cost and most notably, the number of distance computations applied. In applications where the distance measure is computationally expensive (e.g., protein-protein interaction, DNA sequences), the cost of distance computations is the most significant performance index, dominating the overall processing cost. Note that this problem poses some non-trivial challenges such as: (i) objects are defined in a metric space, (ii) the dominance relationship is defined in a dynamically generated set of attributes, (iii) progressive computation is desirable.

4. ALGORITHMIC TECHNIQUES

In this section, we present algorithms for metric-based top-k dominating queries. Initially, we discuss briefly some basic issues regarding the indexing mechanisms and some important properties and then we study the details of the algorithms.

4.1 Indexing Issues

One of the most important factors that may affect performance is the indexing scheme used. Since we focus on metric spaces, metric-based access methods may be employed. Among the available metric-based access methods studied in the literature [5, 2, 3, 4, 13] we select the M-tree [8] which is well appreciated due to its simplicity, its resemblance to the B-tree, its excellent performance and its ability to handle dynamic data sets (i.e., insertions and deletions). However, our methods are orthogonal to the indexing scheme used, as long as incremental k-nearest-neighbor queries are supported.

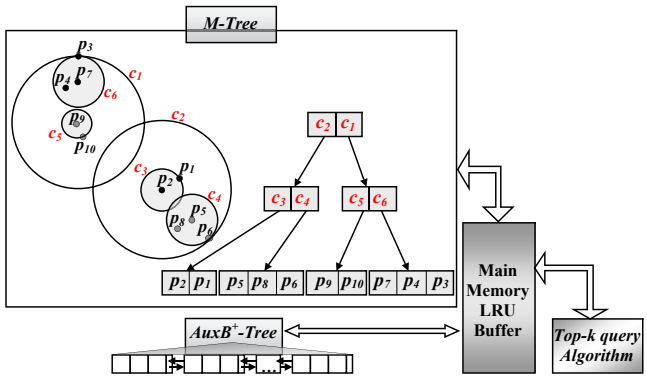


Figure 2: Access methods employed.

Figure 2 depicts the access methods utilized by the studied techniques. In addition to the M-tree, an auxiliary B$^+$-tree (denoted as $AuxB^+$-tree) is being used, which serves as a temporary index for intermediate computations. Each record contains the object ID and specific counters that keep the current cardinalities of intermediate set calculations such as the number of times that an object retrieved during scanning, a clone counter used for exact score computation during backward scanning, its current max-rank position in the nearest neighbor order from the query objects. During execution, the scanned object ID's are inserted into the $AuxB^+$-tree and their corresponding counters are updated. The usefulness of this structure will be clarified later. Both the M-tree and the $AuxB^+$-tree are supported by an LRU buffer which reduces I/O cost due to locality of references.

4.2 The Skyline-Based Algorithm (SBA)

The first algorithm we study is based directly on the observation of [18] that the top-1 object of a monotone scoring function belongs to the skyline.

LEMMA 1. *The top-1 dominating object is always a metric skyline object, i.e., $MSD(Q, 1) \subseteq MSS(Q)$.*

PROOF. The top-1 dominating object t has the maximum dominance score in D, i.e., $dom(t, Q) \geq dom(r, Q), \forall r \in D$. Object t cannot be dominated by any other object x, because in that case x would have a greater dominance value than t ($x \prec t \Rightarrow dom(x) > dom(t)$) which does not hold. Therefore t, is definitely a skyline object. □

The concept of the *Skyline-Based Algorithm* (`SBA`) is very simple: to compute the skyline S of D, we determine the top-1 dominating object $TopObject$ in D from S, remove $TopObject$ from D and repeat the same process until all top-k results have been reported.

The outline of `SBA` is given in Algorithm 1. The metric skyline set S (Line 2) is computed using the B^2MS^2 algorithm proposed in [12], which is the state-of-the-art algorithm for general metric-based skyline queries, since it outperforms previously proposed algorithms, such as MSQ [6,

Algorithm 1 SBA(D,Q,k)

Input: D data set, Q query objects, k number of results
Output: the k best objects

```
01. for i ← 1 to k
02.     S ← MSS of D with respect to Q using B²MS²;
03.     Max ← 0;
04.     TopObject ← ∅;
05.     for each object o ∈ S
06.         dom(o) ← 0;
07.         for each object p ∈ D, p ≠ o
08.             if (o ≺ p) dom(o) + +;
10.         if (dom(o) > Max)
11.             Max ← dom(o);
12.             TopObject ← o;
15.     report TopObject;
16.     remove TopObject from D;
```

7]. The B^2MS^2 algorithm in our case operates over an M-tree index. The objects of the skyline set S and their corresponding dominance values are kept and updated into the $AuxB^+$-tree. Therefore, both random and sorted accesses are supported.

`SBA` reports the top-k results in a progressive manner. However, this method has two important limitations: (i) it performs many unnecessary score computations, since the skyline is often larger than k and (ii) when there is a large number of query objects ($|Q|$), the skyline grows significantly and in some cases approaches the data set cardinality $|D|$. These characteristics may lead to significant performance degradation due to increased processing costs.

4.3 The Aggregation-Based Algorithm (ABA)

The next algorithm we study uses the concept of *aggregation*. The following lemma shows the relationship between the results of a sum-aggregate query to those of a metric top-k dominating query.

LEMMA 2. *If an object p dominates another object r then p will be returned before r in the result list of a top-h aggregate nearest neighbor query ($ANN(Q,h)$) by using the sum-aggregate function.*

PROOF. Let p and r two objects such that $p \prec r$. Then we have:

$$\forall q_i \in Q, d(p,q_i) \leq d(r,q_i) \wedge \exists q_j \in Q, d(p,q_j) < d(r,q_j)$$

By summing those $m = |Q|$ inequalities, we derive:

$$\sum_{i=1}^{m} d(p,q_i) < \sum_{i=1}^{m} d(r,q_i) \Leftrightarrow adist(p,Q) < adist(r,Q)$$

Therefore, p has a strictly smaller aggregate distance than r and it is located before r in the aggregate nearest neighbor query results. □

Lemma 2 associates a top-k dominating query with a top-h aggregate nearest neighbor query using the sum-aggregate function. More specifically, the result set of a top-k dominating query is always a subset of the result set of a top-h aggregate nearest neighbor query (for a sufficiently large h such that $h \geq k$), i.e., $MSD(Q,k) \subseteq ANN(Q,h)$. This is useful for defining a specific search region around the query objects to retrieve candidate objects for the top-k dominating results.

In order to better show the usefulness of Lemma 2, Figure 3 depicts a simple example in the 2-dimensional Euclidean space. There are two query objects q_1, q_2 and let p be an object of the data set. Each circle is centered at a query point and the associated radius corresponds to the distance $d(p,q_1)$ and $d(p,q_2)$ respectively. It is already known from [20] that their intersection area is called the *dominator region* of p, and contains all points that dominate p. Moreover, all points that are dominated by p lie outside the area of both circles, which is called the *dominance region* of p. On the other hand, all objects that have smaller sum-aggregate distances from Q lie into an elliptic area which is defined by the query points $q1, q2$ and intersects point p. The ellipsis crosses the intersection points of the previous two circles, thus the corresponding elliptic area contains the dominator region of p. This suggests that all points which dominate p have smaller sum-aggregate distances than p.

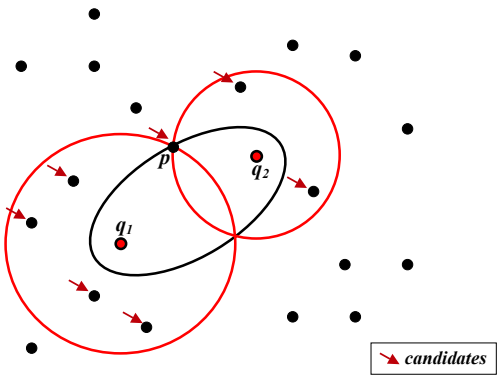


Figure 3: Example for `ABA` description.

In the case where we have more query points the shapes of these regions become more complicated and more difficult to compute. Details for some interesting cases can be found in [19, 20]. The main difficulty is that we do not know in advance a convenient value of h such that $MSD(Q,k) \subseteq ANN(Q,h)$ to answer a top-k dominating query. The following lemma (the corresponding proof is omitted) will help in our quest.

LEMMA 3. *The first sum-aggregate nearest-neighbor of Q is always a metric space skyline object, i.e., $ANN(Q,1) \subseteq MSS(Q)$.*

Assume that we have at hand a convenient value of h such that $MSD(Q,k) \subseteq ANN(Q,h)$. Under this assumption, we may first execute a top-h sum-aggregate nearest-neighbor query and then select the k objects with the highest dominance scores. However, this is not feasible because such a value of h is not known at the beginning of query execution. Thus, an alternative methodology is being followed. Initially, the top-1 sum-aggregate nearest-neighbor object p of Q is computed, i.e., $p = ANN(Q)$. Since p is a skyline object, there is no object that dominates p. Therefore, there are no objects inside p's dominator region and additionally p dominates all other objects lying into its dominance region. This ensures that the top-1 dominating object cannot lie into the dominator/dominance regions of p. Thus, we should search for the top-1 object in the rest of the data set. Candidates are collected in set C by performing simple range queries centered at the query objects $q_1, q_2, ..., q_m$ with radius $d(p,q_1), d(p,q_2), ..., d(p,q_m)$ respectively. Then, we compute the dominating scores of all objects in C and we determine the top-1 dominating object $TopObject$. Next, we remove the $TopObject$ from D and we repeat the same process until all top-k results are reported. The outline of `ABA` is given in Algorithm 2.

In order to have a better view of those candidates let us consider the example of Figure 3. The candidates are the objects contained inside the circles with centers the query points and radii their corresponding distances from p. In our case, there are no objects inside the circle intersection area (dominator region), as p cannot be dominated. Moreover, there are no objects inside the elliptic area, since p is the first sum-aggregate nearest-neighbor of Q.

For the aggregate nearest neighbor query $ANN(Q)$ of Line 2 we use the MBM algorithm of [19] which is the state-of-

Algorithm 2 ABA(D,Q,k)

Input: D data set, Q query objects, k number of results
Output: the k best objects

```
01. for i = 1 to k
02.     p ← 1st sum-aggregate NN of Q using MBM;
03.     C ← ∅;
04.     for each query object q_j ∈ Q
05.         R ← range query from q_j with radius d(p, q_j);
06.         C ← C ∪ R;
08.     Max ← 0;
09.     TopObject ← ∅;
10.     for each object o ∈ C;
11.         dom(o) ← 0;
12.         for each object x ∈ D, x ≠ o
13.             if (o ≺ x) dom(o) + +;
15.         if (dom(o) > Max)
16.             Max ← dom(o);
17.             TopObject ← o;
20.     report TopObject;
21.     remove TopObject from D;
```

the-art algorithm for ANN queries with the sum-aggregate function. The main difference is that we implemented the MBM method to manage M-tree nodes instead of R-tree nodes supported by the original proposal. Range queries (Line 5) are efficiently supported by the M-tree structure. The candidate objects of the set C and their dominance values are kept and updated into the $AuxB^+$-tree.

`ABA` reports the top-k results in a progressive manner. It benefits from the fact that in most cases it is expected that the cost of the ANN query of Line 2 plus the cost of the simple range queries of Line 5, is lower than the cost of a complete skyline computation (as performed by `SBA`). The limitations of `ABA` are as follows: (i) it recalculates up to k times the dominance values of some nearest neighbor candidate objects of C, (ii) when the cardinality of Q increases we must perform a large number of range queries (Line 5), which deteriorates the performance of the algorithm, and (iii) when the query objects are far from each other, the range queries may return a large number of candidates, thus C grows significantly leading to high computational costs.

4.4 The Pruning-Based Algorithm (PBA)

Based on the significant limitations of `SBA` and `ABA`, we center our attention to a third alternative. The key idea behind the Pruning-Based Algorithm (`PBA`) is to incrementally retrieve the nearest neighbors of the query objects in a round-robin fashion, to compute the dominance scores of common neighbors, and, under certain conditions, to extract the top-k results in a progressive manner. We differentiate between two variations, `PBA1` and `PBA2`, that will be described after the presentation of the general idea. Briefly, we use the following techniques which contribute significantly to the reduction of CPU time, I/O cost and the number of distance computations performed:

- alternative techniques are used to compute the score of an object, resulting in more efficient processing, and
- effective pruning is employed, toward eliminating objects that cannot be part of the answer.

In particular, we exploit the following result:

LEMMA 4. *When we have more than one query objects $q_1, ..., q_m$ ($m = |Q| > 1$) and p has been found as the nearest neighbor of all query objects (not necessary with the same nearest neighbor order position), then p dominates all the following nearest neighbors that have not yet been seen in the nearest neighbor order from any query object.*

PROOF. Let $p_{j1}, p_{j2}, ..., p_{jn}$ be the nearest neighbor order of the n objects from the query object q_j, $\forall j = 1, ..., m$. First, we examine the case that these orderings are unique (without equal distances from the query objects), i.e.

$$d(q_j, p_{j1}) < d(q_j, p_{j2}) < ... < d(q_j, p_{jn}), \forall j = 1, ..., m \quad (1)$$

If p has been found as the k_j-th nearest neighbor of q_j, $\forall j = 1, ..., m$, and x is any not found yet object, then: $p = p_{jk_j}, \forall j = 1, ..., m$ and x is an object p_{ji} where $i > k_j, \forall j = 1, ..., m$. Then, using inequality 1 we have: $d(q_j, p_{j1}) < ... < d(q_j, p_{jk_j}) = d(q_j, p) < ... < d(q_j, p_{ji}) = d(q_j, x) < ..., \forall j = 1, ..., m$. Therefore, $d(q_j, p) < d(q_j, x), \forall j = 1, ..., m$, thus p dominates x. In case we have equal distances:

$$d(q_j, p_{j1}) \leq d(q_j, p_{j2}) \leq ... \leq d(q_j, p_{jn}), \forall j = 1, ..., m$$

the object p again dominates x, as x belongs to the next nearest neighbors and not to the objects which have: $d(q_j, p) = d(q_j, x), \forall j = 1, ..., m$. □

Some necessary notations follow: (i) we denote by o_{j1}, o_{j2}, ... , o_{jn} the nearest neighbor order of the n objects from query object q_j, $\forall j = 1, ..., m$. We have: $d(q_j, o_{j1}) \leq d(q_j, o_{j2}) \leq ... \leq d(q_j, o_{jn}), \forall j = 1, ..., m$, (ii) we denote by $rank(o_i, q_j)$ the *rank position* of the object o_i in the nearest neighbor order from the query object q_j, $\forall i = 1, ..., n$, $\forall j = 1, ..., m$ and (iii) we say that two objects x and y are *equivalent* when they have the same distances from the query objects.

The following lemma offers an upper bound for the exact dominance score of a retrieved object o_i. We use this bound as an estimation of the dominance score of o_i denoted as $estdom(o_i)$.

LEMMA 5. *If an object o_i has been retrieved as the $(r_{i,j})$-th nearest-neighbor of query object q_j, where $r_{i,j}$=$rank(o_i, q_j)$, then:*

$$dom(o_i) \leq n - \max_j rank(o_i, q_j) + eq(o_i)$$

PROOF. The object o_i cannot dominate all objects located before its rank position in the nearest neighbor order from the query object q_j. Since this holds for the nearest neighbor order from any query object, it holds also for the order from the query object which maximizes the rank position of o_i. Therefore, $\max_j rank(o_i, q_j)$ objects cannot be dominated by o_i (including itself). Moreover, o_i cannot dominate all its equivalent objects $eq(o_i)$. But, as the equivalent objects of o_i may lie in neighbor order positions before or after o_i, they may be already included into the $\max_j rank(o_i, q_j)$ counted objects. Therefore, o_i may dominate at most $n - (\max_j rank(o_i, q_j) - eq(o_i))$ objects. □

`PBA` uses incremental nearest neighbor retrieval, which is efficiently supported in the M-tree implementation of [8].

Algorithm 3 PBA(D,Q,k)

Input: D data set, Q query objects, k number of results
Output: the k best objects

```
01. H = ∅
02. for i ← 1 to k
03.     do
04.         if (H ← ∅)
05.             call NextCommonNeighbor(D, Q, H);
06.         call NextCommonNeighbor(D, Q, H);
07.         o_a ← H.top;
08.         H.deheap(o_a);
09.         o_b ← H.top;
10.         if (o_a has an estimated dominance score)
11.             dom(o_a) ← ExactScore(o_a, D, Q, H);
12.         if (dom(o_a) < dom(o_b) ∨ dom(o_a) < estdom(o_b))
13.             H.enheap(o_a);
14.     while (dom(o_a) < dom(o_b) ∨ dom(o_a) < estdom(o_b))
15.     report o_a as the top-i dominating object;
16. end for
```

Incremental retrieval is performed by using all query objects in a round robin fashion (i.e., 1st NN from q_1, ... , 1st NN from q_m, 2nd NN from q_1, ... , 2nd NN from q_m, etc.). This idea was first proposed in the *Threshold Algorithm* of [11], and used in several other problems as for example: in distributed skyline queries [1], in aggregate nearest neighbor queries [19], etc.

Using this round-robin incremental retrieval, every time a *common neighbor* object o is detected, it is inserted into a maxheap data structure (H) along with its estimated dominance score ($estdom(o) = n - \max_j rank(o, q_j) + eq(o)$). The heap is prioritized according to the dominance scores of the stored objects (either estimated or exact). It is important to note that H maintains only the common neighbor objects determined so far.

The first two common neighbors are retrieved and inserted into the heap, then the current top object is deheaped and its exact dominance score is calculated (if not available). After that, and before the extraction of the next candidate object from the heap and its score calculation, we always detect and insert into the heap the next common neighbor object with its estimated score. Therefore, there will be always inside the heap a common neighbor object with an estimated score greater than or equal to all subsequent estimated scores. The next result suggests how the current common neighbor object should be handled.

LEMMA 6. *If o_a and o_b are the top-2 common neighbor objects into the heap, and o_a has an exact dominance score $dom(o_a)$ such that:*

$$dom(o_a) \geq estdom(o_b) \text{ or } dom(o_a) \geq dom(o_b)$$

then o_a can be immediately reported as the next top dominating object.

PROOF. This follows from the fact that any next retrieved common neighbor object (which is a candidate for the top-k dominating results), will have a smaller estimated (and subsequently exact) dominance value. □

The result of Lemma 6 provides the progressive behavior of PBA. If the current examined common neighbor object o_a satisfies the condition of Lemma 6, then it is reported as the next top-i dominating query result. Otherwise, o_a is reinserted into the heap H with its exact score and the process is repeated until all top-k results are reported.

Procedure 1 NextCommonNeighbor(D, Q, H)

```
01. do
02.     select the next query object q_j;
03.     o ← NextNearestNeighbor (q_j);
04.     insert (if not) and update counters of o in AuxB+;
05. while o has not been retrieved from all query objects;
06. compute number of equivalent objects of o, eq(o);
07. estdom(o) ← n − max_j rank(o, q_j) + eq(o);
08. H.enheap(o);
```

The outline of PBA is depicted in Algorithm 3. It is important to note that the retrieved objects o along with other useful information (e.g., the number of times retrieved from the query objects ($q_{counter}$), its current max-rank value etc.) are inserted into the $AuxB^+$-tree (Line 4 of Procedure 1) . Therefore, all required intermediate calculations are kept on disk. The only memory resident data structure is the heap H which stores the ID's of the retrieved common neighbor objects determined so far along with their corresponding dominance values (exact or estimated).

4.4.1 Reducing the cost of score computation

The computation of the exact dominance score of an object (Procedure ExactScore of Line 10 in Figure 3) can be performed using the process of the previous algorithms (i.e., Lines 6-9 of SBA outline, Lines 11-14 of ABA outline). However, we can apply a more efficient score computation method which is based on *reverse scanning* [22]. Note that [22] works with a multidimensional space and therefore reverse scanning must be adapted in our case to work with a metric space. The algorithm that uses this type of computation is termed PBA1.

Let U_j be the set of all retrieved nearest neighbor objects before o for the query object q_j, which have distances strictly smaller than o: $U_j = \{x \in D : d(q_j, x) < d(q_j, o)\}$. Let also U be the union of the U_j sets defined by all the query objects $q_j, \forall j = 1, ..., m$: $U = \bigcup_{j=1}^{m} U_j$. The following guarantees the computation of the exact score of o:

LEMMA 7. *For any common neighbor object o with a union set U and a number of equivalent objects $eq(o)$, its exact dominance value is computed using the formula:*

$$dom(o) = n - |U| - eq(o) - 1$$

PROOF. It follows from the fact that o cannot dominate the following objects: (i) any object $x \in U$, as it holds $d(q_j, x) < d(q_j, o)$ for some q_j, (ii) its equivalent objects and (iii) itself. □

This result is important as it enables the computation of the exact dominance score of an object o by just counting the already retrieved objects located before o in the nearest neighbor order from the query objects and by counting the number of its equivalent objects. As all these objects are already retrieved and inserted into the $AuxB^+$-tree,

Procedure 2 ExactScore-RS(o, D, Q, H)

01. $|U| \leftarrow |AUX|$ (initial cardinality of U)
02. **for each** query object $q_j \in Q$
03. **do**
04. $x \leftarrow$ PreviousNearestNeighbor(q_j);
05. **update** counters of x in $AuxB^+$-tree;
06. **if** ($qc_{counter}$ of $x = 0$) delete x from U;
07. **while** $d(q_j, x) \geq d(q_j, o)$
08. **end-for**
09. $dom(o) = n - |U| - eq(o) - 1$;
10. **return** $dom(o)$;

the required counting can be successfully performed inside the $AuxB^+$-tree, without materializing the sets U_j and U. Moreover, the formula of Lemma 7 requires only the cardinality of the set U and not its contents. Let us describe this process in more detail:

Inside ExactScore-RS, object o has already been retrieved from all query objects (it is a common neighbor object), and due to the round-robin scan it may have been retrieved for some query objects in earlier rank positions than the last one. Therefore, inside the $AuxB^+$-tree, some objects located after o in the nearest neighbor order may have been counted ($q_{counter}$) more than once. To correct the counting we scan backwards in the corresponding nearest neighbor orders until we find an object x such that $d(q_j, x) < d(q_j, o)$. Then, we update some additional counters of the retrieved objects in $AuxB^+$-tree. More specifically, we use a copy of the $q_{counter}$ of x (the $qc_{counter}$), which is updated with the total number of the retrievals of x before the execution of the Compute-Exact-Score procedure, minus the total number of the retrievals of x during the reverse scanning. Every time the $qc_{counter}$ of an object x becomes zero, that object must be excluded from the final union set U.

The outline of the score computation method using the reverse scanning technique is given in Procedure 2. Note that the number of equivalent objects of o has already been computed (Line 6 of Procedure 1) and the corresponding objects have been inserted in the heap H (Line 8 of Procedure 1). For the following discussion, let AUX be the set containing all unique objects inserted into the $AuxB^+$-tree.

The exact score computation can be also performed by utilizing information already stored in the $AuxB^+$-tree structure. We denote by $Lpos_{o_i}(q_j)$ the minimum rank position of the objects in the nearest neighbor order from q_j, which have an equal distance with o_i, i.e., $Lpos_{o_i}(q_j) = \min_h\{rank(o_h, q_j) : d(q_j, o_h) = d(q_j, o_i)\}$. It is important to note that the $Lpos$ rank positions are recorded into the $AuxB^+$-tree during the incremental nearest neighbor retrieval together with the q counters, thus they are available before start calculating the exact dominance score of o. However, there is a significant difference between the common neighbor objects that have been detected so far (including o) and all other objects that have already been inserted into the $AuxB^+$-tree: the first group of objects has all their $Lpos$ rank positions values already recorded into the $AuxB^+$-tree, while for the second group only some of their $Lpos$ rank positions values have been recorded (as they have been detected only from some query objects q_j).

Procedure 3 ExactScore-AUX(o, Q)

01. $dom_{in} = 0$
02. read all equal-distance groups of o (if still not read)
03. **for each** object $o_i \in AuxB^+$-tree
04. ff = true
05. **for each** query object $q_j \in Q$
06. **if** ($Lpos_{o_i}(q_j) \neq$ NULL) $\wedge$ ($Lpos_{o_i}(q_j) < Lpos_o(q_j)$)
07. $ff \leftarrow$ false; break;
08. **end-for**
09. **if** (ff = true)
10. $fe \leftarrow$ true
11. **for each** query object $q_j \in Q$
12. **if** ($Lpos_{o_i}(q_j) \neq Lpos_o(q_j)$)
13. $fe \leftarrow$ false; break;
14. **end-for**
15. **if** (fe = true) $ff \leftarrow$ false
16. **end-if**
17. **if** (ff = true) $dom_{in}++$;
18. **end-for**
19. $dom(o) \leftarrow dom_{in} + n - |AUX|$;
20. **return** $dom(o)$;

The above observation leads to a second alternative for the exact score computation, which is outlined in Procedure 3. The pruning-based algorithm that uses Procedure 3 for the exact score computation is denoted as `PBA2`. Recall that `PBA1` and `PBA2` differ only on the way exact scores are computed.

4.4.2 Applying effective pruning

The methodology of algorithms `PBA1` and `PBA2` enable the usage of several pruning heuristics, which reduce the runtime costs further. We propose three different types of pruning heuristics: (i) *Discard Heuristics - DH*, which can discard objects that have not been retrieved yet, (ii) *Early Pruning Heuristics - EPH*, which can prune objects before the calculation of their exact dominance scores and (iii) *an Internal Pruning Heuristic - IPH*, which can prune objects during the procedure of the exact dominance score calculation.

Some of the proposed heuristics use a global pruning value G for the dominance scores. G is initialized to 0, and it is updated from the first time that we have calculated the exact dominance scores of k common neighbor objects (which have been inserted into the heap H), and after any change of the current exact top-k dominating object o_k in the heap during the retrieval of the next common neighbor objects. G is always updated with the exact dominance score of the current exact top-k dominating object minus 1, i.e., $G = dom(o_k) - 1$. Any object with an exact dominance value smaller than or equal to G can be pruned safely.

Discard Heuristic $DH1$: If o_k is the current exact top-k dominating object in the heap H, U_k is its calculated union set, and $\{eq(o_k)\}$ is the set of its equivalent objects, then all objects of the set: $D - \{U_k \cup \{eq(o_k)\} \cup \{o_k\}\}$ can be discarded (pruned before their retrieval).

Discard Heuristic $DH2$: If o_p is an object which has been pruned by any other heuristic, U_p is its calculated union set, and $\{eq(o_p)\}$ is the set of its equivalent objects, then all objects of the set: $D - \{U_p \cup \{eq(o_p)\} \cup \{o_p\}\}$ can be discarded.

Discard Heuristic $DH3$: The first time that we have calculated the exact dominance scores of k objects into the

heap, all objects that have not been inserted yet into the $AuxB^+$-tree can be discarded.

Early Pruning Heuristic $EPH1$**:** If o_k is the current exact top-k dominating object in the heap H, o is the current retrieved common neighbor object for calculations, and $o_k \prec o$, then o can be pruned.

Early Pruning Heuristic $EPH2$**:** If o_i is any exact calculated dominating object that is after the exact top-k dominating object into the heap H, o is the current retrieved common neighbor object for calculations, and $o_i \prec o$, then o can be pruned.

The following heuristics require some additional notations. We denote by $Lpos_o(q_j)$ the rank position of the leftmost object o_i in the nearest neighbor order from q_j, which has an equal distance to o, i.e., $d(q_j, o_i) = d(q_j, o)$. We also denote by pos the maximum rank position retrieved so far, i.e., $pos = \max_j rank(o, q_j)$, where o is the last retrieved common neighbor object. Finally, we denote by a the index of the last selected query object from the round-robin scan ($a = j$, if we are in the query object q_j).

Early Pruning Heuristic $EPH3$**:** If o is the current retrieved common neighbor object for calculations, then o can safely be pruned if the following holds:

$$n - \max_j Lpos_o(q_j) \leq G$$

Early Pruning Heuristic $EPH4$**:** If o is the current retrieved common neighbor object for calculations, then o can be pruned if the following holds:

$$n - |AUX| - 1 + \sum_{j=1}^{m} [pos - Lpos_o(q_j)] + a \leq G$$

Early Pruning Heuristic $EPH5$**:** If o_i is any exact calculated dominating object into the heap H, o is the current retrieved common neighbor object for calculations, then o can be pruned if the following holds:

$$dom(o_i) + eq(o_i) + \sum_{\substack{j: Lpos_{o_i} \\ > Lpos_o}} [Lpos_{o_i}(q_j) - Lpos_o(q_j)] \leq G$$

Internal Pruning Heuristic IPH**:** If o is the current retrieved common neighbor object, and during calculations the following holds:

$$N - |AUX| + curDom(o) + \sum_{j=1}^{m} |curPos(q_j) - Lpos_o(q_j)| \leq G$$

(where $curDom$ is the current recorded score of o and $curPos$ is the current rank position in the nearest neighbor order from q_j during the reverse scan), then o can safely be pruned, and the remaining calculations can be skipped.

Early and internal pruning rules are applied to the heap objects, whereas all discard pruning rules are applied to both heap and $AuxB^+$-tree objects. This process results in eliminating a significant number of objects.

5. PERFORMANCE EVALUATION

In this section, we present experimental results demonstrating the performance of the studied algorithms `SBA`, `ABA`, `PBA1` and `PBA2`. We exclude the brute-force algorithm from all results, because its performance is several orders of magnitude worse than that of the other algorithms. All algorithms have been implemented in C++ and all experiments have been conducted on a Pentium IV@3GHz machine. The disk page size is set to 4KB for all access methods and a cost of 8msec is attributed to each page fault. An LRU buffer has been used to absorb page faults. Two separate buffers have been used: one for the M-tree access method (10% of M-tree size) and one for the rest of the access methods (20% of db size).

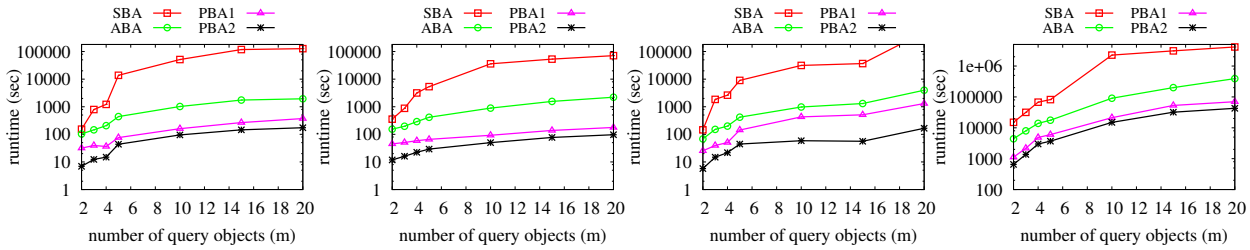
We have used both real-life and synthetic data sets. The data sets used are briefly described below:

- The FOREST COVER (FC) data set [1] contains 581,012 forest land cells containing 55 attributes. The first ten numeric attributes have been used, representing positions, distances to roads, hydrology etc. The distance function used is the Euclidean distance.
- The ZILLOW (ZIL) data set [2] contains real estate data used also in [23]. It contains more than 2M records, but we selected the 1,224,406 records with non zero or empty values in all their attributes. The data set has 5 attributes with the following order: number of bathrooms, number of bedrooms, living area, price, lot area. The Euclidean distance has been used as the distance function.
- The CALIFORNIA (CAL) road network [3] contains 1,965,206 nodes and 5,533,214 edges. The average node degree is equal to 2.55, the average edge weight is 8.78 and diameter (maximum shortest path distance) is 16,828.54. The distance function used is the shortest path between network nodes.
- For comparison purposes, we have also used a synthetic data set (UNI), which contains 1,000,000 4-dimensional data objects with attribute values respecting uniformity and independence. The distance used is the Manhattan (L_1) distance.

The values reported are averages from 20 different executions of the algorithms, using randomly chosen query objects. Query objects are selected from the data set D according to the parameter c which gives the *coverage* of the query set Q. More specifically, the coverage is defined as the ratio of the minimum radius required to enclose all query objects in Q over the minimum radius required to cover the whole data set. The larger the c value the more distant the query objects contained in Q. Unless otherwise specified, the parameters take the following values: (i) number of query objects (m) ranges between 2 and 20 with a default value of 5, (ii) query coverage (c) takes the values 1%, 5%, 10%, 20%, 30%, 50%, 100% with a default value of 20% and (iii) the number of results (k) ranges between 1 and 30 with a default value of 10.

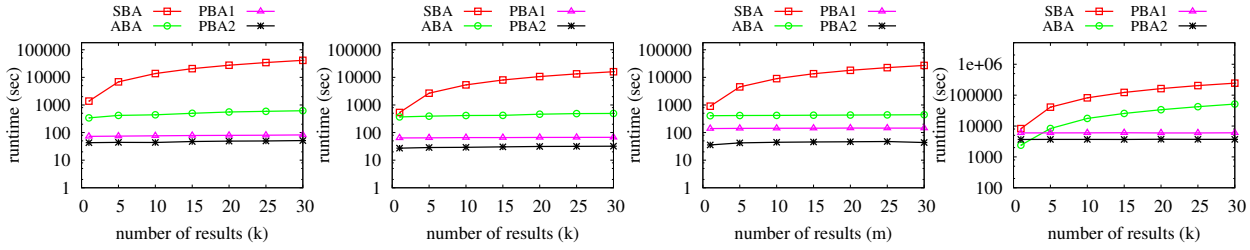
There are three basic criteria used to evaluate and compare the performance of the proposed algorithms: (i) the CPU time required for computations, (ii) the I/O time devoted for disk accesses and (iii) the number of distance computations required. It is important to note that for top-k dominating queries with $k > 1$, due to the progressiveness property of the proposed algorithms, any top-i result with $i < k$ will be reported earlier. This behavior enables the

[1] http://kdd.ics.uci.edu
[2] http://www.zillow.com
[3] http://snap.stanford.edu/data/index.html



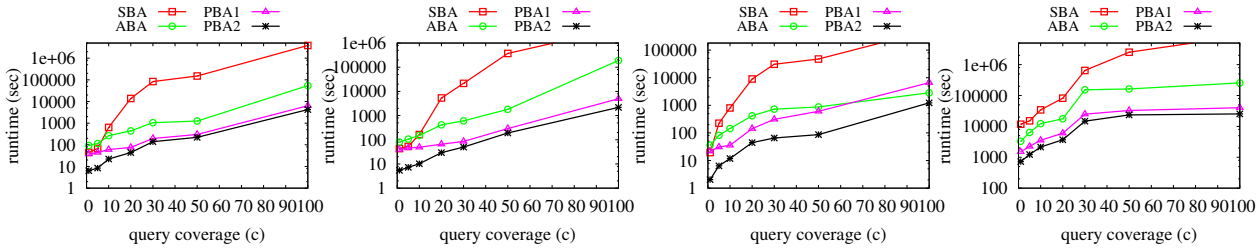
(a) UNI (b) FC (c) ZIL (d) CAL

Figure 4: Running time vs. number of queries m.



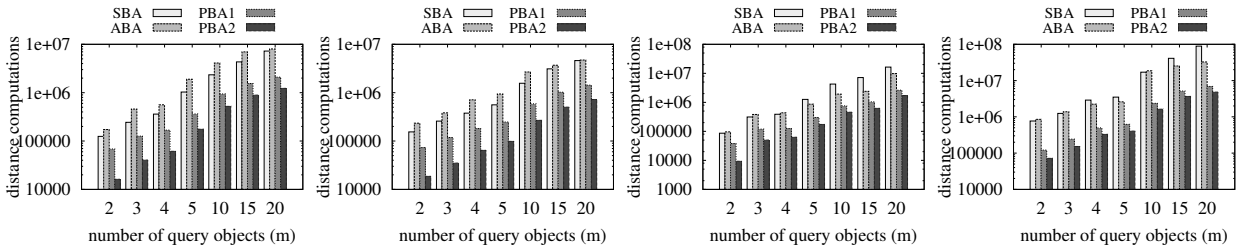
(a) UNI (b) FC (c) ZIL (d) CAL

Figure 5: Running time vs. number of results k.



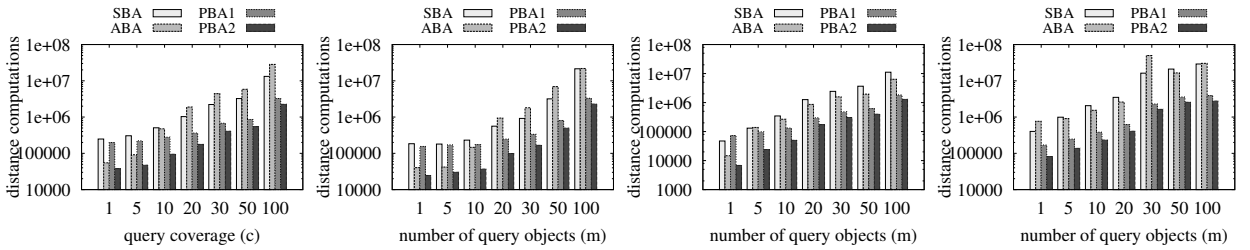
(a) UNI (b) FC (c) ZIL (d) CAL

Figure 6: Running time vs. query coverage c.



(a) UNI (b) FC (c) ZIL (d) CAL

Figure 7: Number of distance computations vs. number of query objects m.



(a) UNI (b) FC (c) ZIL (d) CAL

Figure 8: Number of distance computations vs. query coverage c.

Data		number of query objs (m)					number of results (k)				query coverage (c)		
		2	**5**	**10**	**15**	**20**	**5**	**10**	**20**	**30**	**1%**	**10%**	**20%**
UNI	CPU	0.18	11.60	52.52	94.96	125.01	11.12	11.61	13.84	15.32	0.44	3.25	11.61
	I/O	6.77	32.22	44.84	50.34	48.50	32.97	32.22	35.21	35.97	5.93	18.92	32.22
FC	CPU	0.24	2.83	12.54	30.58	47.34	2.65	2.82	3.32	3.62	0.21	0.43	2.83
	I/O	11.62	26.43	37.54	46.74	49.63	26.09	26.43	28.07	28.43	5.24	9.76	26.43
ZIL	CPU	0.05	7.54	16.94	17.99	49.64	5.50	7.54	9.41	11.34	0.03	0.46	7.54
	I/O	5.71	36.89	41.83	38.03	115.85	36.87	36.89	36.91	32.25	2.01	11.33	36.89
CAL	CPU	624.52	3637.64	14828.23	31810.36	42595.36	3627.67	3637.64	3669.07	3646.63	714.01	2111.09	3637.64
	I/O	26.00	47.62	140.66	195.28	195.47	59.36	47.62	59.37	59.38	11.34	32.07	47.62

Table 2: CPU and I/O cost (in seconds) for PBA2.

Data	number of query objs (m)					number of results (k)				query coverage (c)			
	2	**5**	**10**	**15**	**20**	**5**	**10**	**20**	**30**	**1%**	**10%**	**20%**	**50%**
UNI	15	16	16	21	24	11	13	29	47	10	15	13	19
FC	14	15	16	16	16	7	14	29	39	12	12	14	20
ZIL	16	115	148	182	50	80	115	164	201	12	21	115	41
CAL	253	272	45	51	51	224	272	312	333	263	87	272	275

Table 3: Number of exact score computations for algorithms PBA1 and PBA2.

retrieval of the first results without the need for waiting the computation of the complete answer set.

First, we compare the proposed algorithms by varying the cardinality of the query objects m and by keeping the other parameters to their default values. Figure 4 depicts the performance results. We observe that as we increase the cardinality, the performance cost increases. SBA generally does not perform well in comparison to ABA, PBA1 and PBA2 but it performs better than ABA in uniform data and small coverage. This is because the minimum enclosing ball of the query set (which the algorithm B^2MS^2 uses) and the uniformity of the metric space produce upper bounds that can prune a significant amount of objects. ABA performs better than SBA in real-life data and in larger Q covering areas. PBA2 outperforms the other algorithms in all cases.

In the following experiment series we compare the algorithms, by varying the number of the top results k. Figure 5 depicts the performance results. We observe that as we increase k, SBA and ABA significantly increase their cost because their outer loop forces them to make many re-calculations using the same objects. Again, PBA2 outperforms the other algorithms significantly.

In the next experiment series we compare the proposed algorithms, by varying the query coverage c and by keeping the other parameters to their default values. Figure 6 depicts the performance results. We observe that as we increase the parameter c (producing a spatial anti-correlation), the distances between the query objects become larger, the cardinality of metric space skyline increases and thus, SBA shows the worst performance incurring significant CPU and I/O costs. In contrast, ABA performs better, whereas PBA1 and PBA2 outperform the other algorithms significantly (one to three orders of magnitude).

The number of distance computations invoked by the algorithms are given in Figures 7 and 8 for all data sets. We note that in many applications a single distance computation may be more computationally intensive than several I/O operations. In such a case, the metric function is very expensive and consequently, the number of distance computations affects query response time significantly (being the dominant cost in comparison to I/O time). Thus, it is important to reduce the number of distance computations as much as possible toward efficient query processing. Among the proposed algorithms, PBA2 requires the smallest number of distance computations in all cases. This behavior is attributed to both the pruning rules applied and the way the $AuxB^+$-tree structure is exploited.

In the next series of experiments, we focus on PBA1 and PBA2. In particular, Table 2 reports the CPU and I/O costs for PBA2. For few query points, low query coverage and easy-to-compute distance measures, normally the I/O cost is significantly higher than the CPU cost. However, there are cases where the large number of distance computations in association with more expensive distance measures (e.g., shortest path distance) leads to a significant increase in the CPU time, dominating the total cost. In Table 2, this is shown by the highlighted CPU costs, indicating that I/O cost is insignificant in comparison to CPU time. This is the main reason why reducing the number of distance computations is very important and PBA2 is very successful towards this direction.

Another important factor that is unique in PBA1 and PBA2, is the number of exact score computations performed. Table 3 reports the number of exact score computations for all data sets. It is important to note that in comparison to the data set size there is only a small fraction of exact score computations performed by these algorithms, which is one of the main ingredients for their excellent performance.

In conclusion, the pruning-based algorithms PBA1 and PBA2 perform orders of magnitude better that SBA and ABA and, therefore, are the preferred choices for top-k dominating queries, where attributes are generated dynamically from distances in a metric space. This effect is attributed mainly to the sophisticated pruning mechanisms employed as well as on the underlying data structures used for performance boosting.

6. CONCLUSIONS

Top-k dominating queries combine the advantages of regular top-k and skyline queries, by bounding the size of the result without the need for user-defined scoring functions. This paper contains the first work in metric-based top-k dominating queries, where distances among objects are computed by means of a metric function and attribute values of each object are generated dynamically, based on the distance between the object and a set of query objects.

Four progressive algorithms are studied: the first one (`SBA`) is based on metric skyline computation, the second one (`ABA`) is an extension of the aggregation-based nearest-neighbor technique, whereas the third and forth one (`PBA1` and `PBA2`) use incremental nearest-neighbor search equipped by: (i) a set of effective pruning rules to reduce the search space and (ii) an efficient score computation to reduce runtime. All algorithms operate over any metric-based access method (the M-tree has been used in this work) with the only requirement that incremental nearest-neighbor queries are supported. The performance evaluation study shows that the pruning-based algorithms show the best overall performance in terms of CPU time, I/O cost and number of distance computations, offering runtimes that are between one and three orders of magnitude better than those of `SBA` and `ABA`.

An interesting direction for future work is the study of randomized techniques toward reducing computation time by sacrificing the accuracy of the answer. Another interesting extension is to consider the problem in a parallel/distributed setting, offering additional scalability, especially for massive data sets.

7. ACKNOWLEDGMENTS

This research is supported by a Heraclitus II fellowship, the FP7 INSIGHT and the ARISTEIA MMD projects.

8. REFERENCES

[1] W. T. Balke, U. Guentzer, J. X. Zheng, "Efficient Distributed Skylining for Web Information Systems", *Proc. of the 7th Int. Conference on Extending Database Technology (EDBT)*, pp.256-273, 2004.

[2] T. Bozkaya, M. Ozsoyoglu, "Distance-Based Indexing for High-Dimensional Metric Spaces", *Proc. of the ACM SIGMOD Conference*, pp.357-368, 1997.

[3] T. Bozkaya, M. Ozsoyoglu, "Indexing Large Metric Spaces for Similarity Search Queries", *ACM Transactions on Database Systems*, 24(3), pp.361-404, 1999.

[4] S. Brin, "Near Neighbor Search in Large Metric Spaces", *Proc. of the 21st Int. Conference on Very Large Data Bases (VLDB)*, pp.574-584, 1995.

[5] E. Chávez, G. Navarro, R. Baeza-Yates, J. L. Marroquín, "Searching in metric spaces", *ACM Computing Surveys*, 33(3), pp.273-321, 2001.

[6] L. Chen, X. Lian, "Dynamic skyline queries in metric spaces", *Proc. of the 11th Int. Conference on Extending Database Technology (EDBT)*, pp.333-343, 2008.

[7] L. Chen, X. Lian, "Efficient Processing of Metric Skyline Queries", *IEEE Transactions on Knowledge and Data Engineering*, 21(3), pp.351-365, 2009.

[8] P. Ciaccia, M. Patella, P. Zezula, "M-tree: An Efficient Access Method for Similarity Search in Metric Spaces", *Proc. of the 23rd Int. Conference on Very Large Data Bases (VLDB)*, pp.426-435, 1997.

[9] K. Deng, X. Zhou, H. T. Shen, "Multi-source Skyline Query Processing in Road Networks", *Proc. of the 23rd Int. Conference on Data Engineering (ICDE)*, pp.796-805, 2007.

[10] R. Fadel, K.V. Jakobsen, J. Katajainen, J. Teuholab, "Heaps and Heapsort on Secondary Storage", *Theoretical Computer Science*, 220, pp.345-362, 1999.

[11] R. Fagin, A. Lotem, M. Naor, "Optimal Aggregation Algorithms for Middleware", *Proc. of the ACM Symposium on Principles of Database Systems (PODS)*, pp.102-113, 2001.

[12] D. Fuhry, R. Jin, D. Zhang, "Efficient Skyline Computation in Metric Space", *Proc. of the 12th Int. Conference on Extending Database Technology (EDBT)*, pp.1042-1051, 2009.

[13] G. R. Hjaltason, H. Samet, "Ranking in Spatial Databases", *Proc. of the 4th Int. Symposium on Advances in Spatial Databases (SSD)*, pp.83-95, 1995.

[14] V. Hristidis, N. Koudas, Y. Papakonstantinou, "PREFER: A System for the Efficient Execution of Multi-Parametric Ranked Queries", *Proc. of the ACM SIGMOD Conference*, pp.259-270, 2001.

[15] M. Kontaki, A. N. Papadopoulos, Y. Manolopoulos, "Continuous Top-k Dominating Queries", *IEEE Transactions on Knowledge and Data Engineering*, 2011 (accepted).

[16] X. Lian, L. Chen, "Top-k Dominating Queries in Uncertain Databases", *Proc. of the 12th Int. Conference on Extending Database Technology (EDBT)*, pp.660-671, 2009.

[17] E. A. Leicht, P. Holme, M. E. J. Newman, "Vertex Similarity in Networks", *Physics Review E*, 73, 2006.

[18] D. Papadias, Y. Tao, G. Fu, B. Seeger, "Progressive Skyline Computation in Database Systems", *ACM Transactions on Database Systems*, 30(1), pp.41-82, 2005.

[19] D. Papadias, Y. Tao, K. Mouratidis, C. K. Hui, "Aggregate Nearest Neighbor Queries in Spatial Databases", *ACM Transactions on Database Systems*, 30(2), pp.529-576, 2005.

[20] M. Sharifzadeh, C. Shahabi, "The Spatial Skyline Queries", *Proc. of the 32nd Int. Conference on Very Large Data Bases (VLDB)*, pp.751-762, 2006.

[21] D. Skoutas, D.Sacharidis, A. Simitsis, V. Kantere, T. Sellis, "Top-k Dominant Web Services Under Multi-Criteria Matching", *Proc. of the 12th Int. Conference on Extending Database Technology (EDBT)*, pp.898-909, 2009.

[22] E. Tiakas, A. N. Papadopoulos, Y. Manolopoulos, "Progressive Processing of Subspace Dominating Queries", *The VLDB Journal*, 2011 (accepted).

[23] A. Vlachou, C. Doulkeridis, Y. Kotidis, "Angle-Based Space Partitioning for Efficient Parallel Skyline Computation", *Proc. of the ACM SIGMOD Conference*, pp.227-238, 2008.

[24] M. L. Yiu, N. Mamoulis, "Efficient Processing of Top-k Dominating Queries on Multi-Dimensional Data", *Proc. of the 33rd Int. Conference on Very Large Data Bases (VLDB)*, pp.483-494, 2007.

[25] M. L. Yiu, N. Mamoulis, "Multi-Dimensional Top-k Dominating Queries", *The VLDB Journal*, 18(3), pp.695-718, 2009.

[26] W. Zhang, X. Lin, Y. Zhang, J. Pei, W. Wang, "Threshold-based Probabilistic Top-k Dominating Queries", *The VLDB Journal*, 19(2), pp.283-305, 2010.