Processing Top-k Dominating Queries in Metric Spaces

ELEFTHERIOS TIAKAS, Aristotle University of Thessaloniki
GEORGE VALKANAS, Stevens Institute of Technology
APOSTOLOS N. PAPADOPOULOS and YANNIS MANOLOPOULOS,
Aristotle University of Thessaloniki
DIMITRIOS GUNOPULOS, University of Athens

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 cannot 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 attribute 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 datasets.

Categories and Subject Descriptors: H.2.4 [**Database Management**]: Systems—*Query processing*

General Terms: Algorithms, Performance

Additional Key Words and Phrases: Dominating queries, metric spaces, distance computation

ACM Reference Format:
Eleftherios Tiakas, George Valkanas, Apostolos N. Papadopoulos, Yannis Manolopoulos, and Dimitrios Gunopulos. 2016. Processing top-k dominating queries in metric spaces. ACM Trans. Datab. Syst. 40, 4, Article 23 (January 2016), 38 pages.
DOI: http://dx.doi.org/10.1145/2847524

1. INTRODUCTION

Preference-based queries [Hristidis et al. 2001] 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: (1) *top-k queries* and (2) *skyline queries*.

Top-k query processing has been an active research area spanning disciplines like Web search, p2p-based retrieval, and multimedia databases, to name a few. The query's

Authors' addresses: E. Tiakas, A. N. Papadopoulos, and Y. Manolopoulos, Department of Informatics, Aristotle University of Thessaloniki, 54124 Thessaloniki, Greece; emails: {tiakas, papadopo, manolopo}@csd.auth.gr; G. Valkanas, Stevens Institute of Technology, Howe School of Technology Management, Hoboken, New Jersey, USA; D. Gunopulos, Department of Informatics and Telecommunications, University of Athens, Athens, Greece; email: dg@di.uoa.gr.

DOI: http://dx.doi.org/10.1145/2847524

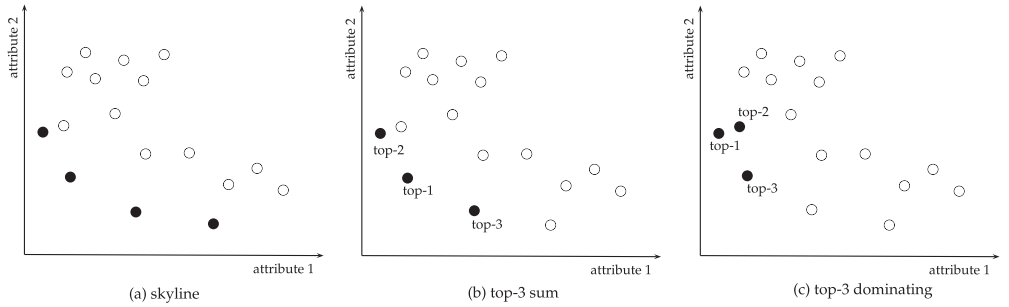

Fig. 1. The three types of preference queries: (a) skyline, (b) top-k, and (c) top-k dominating (for $k = 3$).

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. For example, it is not straightforward to combine the attributes "price" and "screen size" when selecting the laptop that best suits our needs. Moreover, scaling an axis (e.g., from dollars to cents) in general will change the ranking of the objects.

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 than 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), whereas its size depends on the dataset's underlying properties such as the data distribution and dimensionality.

It is not hard 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 Yiu and Mamoulis [2007, 2009]: 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: (1) the result size is controllable, (2) the result is scale invariant, (3) an intuitive dominance-based score is assigned to each object, and (4) no other user-defined scoring function is required.

Figure 1 depicts an example for each of the three query types, that is, skyline queries, top-k queries, and top-k dominating queries for the two-dimensional case (there are only two attributes of interest). For the top-k query, we assume that the scoring function used is the sum of the attribute values.

The importance of this query type has been prominently exemplified by numerous works on the topic and is partly due to its simple yet intuitive explanation: the more points a point p dominates, the better p is. Moreover, it simulates how users select items, in lack of better alternatives: for example, if the "best" camera is out of stock, the second best is picked, and so forth. The query was initially proposed in Yiu and Mamoulis [2007] and enhanced later in Yiu and Mamoulis [2009]. It has been found useful in uncertain databases [Lian and Chen 2009] and in a streaming setting [Kontaki et al. 2012]. Its practicality was also demonstrated in subspace dominance queries [Tiakas et al. 2011], where the computation of dominance relationships is based on a (user-defined) subset of the attributes.

Motivation. The aforementioned techniques operate under the assumption that each object is associated with an attribute vector with fixed values, and that dominance is

based on these attribute values. However, there are domains and areas of application where these techniques cannot be employed, because the fixed-value vector-based representation is impossible or inefficient. Some characteristic examples of these domains include biology, social and road networks, and graphs.

Consider, for instance, a road network that models the distance between points in a city. A food chain is interested in opening a new store in the area but is unsure about the location where to open it. Knowing its customer base, it has identified three distinct subareas that will provide its basic pool of customers. Each subarea can be represented on the road network by its centroid, or the point closest to it. Starting with these three points, the chain wants to identify a set of alternatives where the new store can be situated, so that it is convenient for its clientele to visit and/or deliver to them.

Any possible location for the store can be defined by its distance to the three subareas, $\overline{d} = (d_1, d_2, d_3)$, where the distance is computed on the road network. A location l_1 that is farther from all three centroids as opposed to a location l_2 is dominated by l_2, and is therefore a poorer choice. In simpler terms, l_2 is more convenient to reach than l_1 for *all* three subareas. However, each location is also associated with a price, and perhaps the most convenient of all is not an affordable option. Therefore, the food chain would have to consider the second-best option, perhaps the third, and so on.

Instead of using the sum of distances to rank the alternatives, the chain could use the *dominance* relation and select the top-k dominating locations. To begin with, the overall distance is implicitly captured, as the location with the minimum distance for all three locations will be the top-1. Moreover, ranking by top-k dominance also considers the distance of the alternatives and puts things into perspective. For example, the point with the minimum sum distance—which seems like a good choice, optimizing convenience in a combined way—tells us nothing about the distribution of the other alternatives and could be entirely isolated. On the other hand, a location that dominates 30 alternatives tells us that there are at least 30 other locations distributed in approximately the same area: given that they are dominated points, their distances are constrained in a certain way, placing them (roughly) on the same part of the region that the food chain is considering. The number of alternatives in a region could be indicative of that region's status (e.g., economic growth); therefore, a location that dominates 30 alternatives is not only closer but also most likely in a better-off area compared to a location that dominates only five. Finally, knowing about the number of dominated alternatives, the food chain could now evaluate whether a location is worth the price, and potentially leverage that information as a negotiation tool.

Similar situations may arise in social networks, where we want to recommend new connections. Using the user's profile, we can identify, for instance, the top-5 topics of interest and then extract one representative (e.g., expert) node for each topic from the network. The recommendation engine would then come up with the top-k nodes in the social network that are most similar with these five representatives. Following the same rationale of our previous example, the recommended nodes can be derived through a top-k dominating query, returning those nodes that are consistently more similar with all five representatives, compared to other nodes in the network. That does not necessarily mean that the recommended nodes have the maximum values in *all* dimensions—though, in that case, they would be the top-k result—but rather that for each selected node, its similarity with each of the five representatives is better than a portion of the network that the selected node dominates. Therefore, the top-k dominating query inherently takes into account the underlying data distribution.

We stress that in both examples, the only information we have is the distance between the points in the query set Q and the dataset. The naive approach would be to transform the original data to a $|Q|$-dimensional space, where each point in the original dataset is denoted by its distance to each point in the query set. After this transformation, we

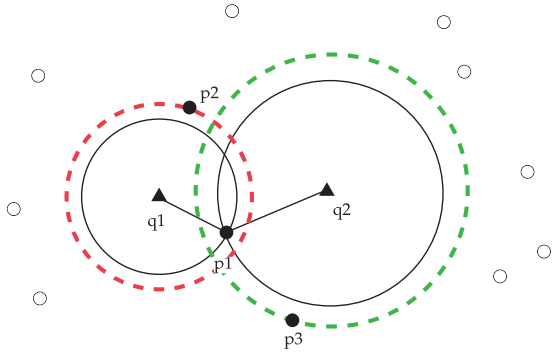

Fig. 2. A top-3 dominating query with two query objects q_1 and q_2.

can apply the top-k dominating query, as described in Yiu and Mamoulis [2007, 2009]. Such a solution, however, would entail a huge transformation overhead, which we must pay for in every query set Q. It is therefore in our best interest to solve the problem in the original space. For these reasons, we study, for the first time, algorithms for top-k dominating queries, where objects are defined in any *metric space* and attributes are based on the distances among data objects and query objects.

Outline of Our Solution. As we will demonstrate throughout our work, top-k dominating queries in metric spaces may be addressed by adapting existing techniques. However, such approaches are highly inefficient, due to a high degree of unnecessary recomputations and increased I/O costs. To overcome these deficiencies, we propose a novel, theoretically sound, and efficient technique, which operates directly on the original (metric) space, without the need for data transformations.

To better understand the inner workings of our solution, consider Figure 2, showing a top-3 dominating query in the 2-dimensional Euclidean space. Two query objects q_1 and q_2 are also depicted. We start by extracting the nearest neighbors of each of the query points, q_1 and q_2. Point p_1 is the first nearest neighbor of both query points, and consequently is the top-1 object of the top-k dominating query. The reason is that no other point lies inside either the circle $C_1(q_1, d(q_1, p_1))$ or $C_2(q_2, d(q_2, p_1))$, meaning that no other point has lower distances from both q_1 and q_2. Therefore, p_1 dominates all other points and its dominance score is $dom(p_1) = 11$.

Since we have not already retrieved the top-3 objects, we proceed by extracting the second nearest neighbor for the two query points. In this case, however, the nearest neighbors are not the same: the second nearest neighbor of q_1 is p_2, whereas that of q_2 is p_3. Since $d(q_1, p_2) < d(q_1, p_3)$ (p_3 lies outside $C_3(q_1, d(q_1, p_2))$) and $d(q_2, p_2) > d(q_2, p_3)$ (p_2 lies outside $C_4(q_2, d(q_2, p_3))$), 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 dashed circles (except p_1). As a result, their dominance score is $dom(p_2) = 9$ and $dom(p_3) = 9$, respectively, whereas the remaining points have a dominance score less than 9. Based on the previous discussion, we conclude that 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.

As explained through our illustrative example, our proposed technique operates in rounds. During each round, we retrieve, for each query object $q_i \in Q$, its next nearest neighbor from the dataset D. We then identify objects that are *common neighbors* across all query objects, and based on their rank position for each q_i, we establish whether the item will be in the top-k result or not. Using the same information of an object's rank position, we can determine whether we should continue extracting more neighbors or

terminate the process. The terminating condition is based on a variation of Fagin's Thresholding Algorithm [Fagin et al. 2001], tailored to our problem's characteristics. We also apply a number of heuristics, using similarly inspired ideas, to reduce the computational and memory footprint of our algorithm.

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 datasets.
(2) We propose efficient techniques for answering metric-based top-k dominating queries, all of which exhibit the *progressiveness* property. In particular, we study two algorithms, the Skyline-Based Algorithm (SBA) and the Aggregation-Based Algorithm (ABA), that are based on the adaptation of existing techniques. However, the most efficient computation is achieved by using the new Pruning-Based Algorithms (PBAs) that use intelligent pruning mechanisms.
(3) We present a detailed performance evaluation of all studied techniques, based on real-life and synthetic datasets, 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.

Extensions Beyond the Conference Version. This article is an extended version of the EDBT 2014 paper [Tiakas et al. 2014]. This journal version contains several enhancements with respect to the conference version. The most significant changes are summarized here:

—We have included two more datasets in the performance evaluation, the San Francisco (SF) road network and the Proteins (PR) dataset. The purpose of SF is to validate the results reported in the conference version for the California (CAL) road network, whereas the purpose of PR is to test the performance of the algorithms when the distance measure is the Levenshtein distance. Thus, we cover the most widely used distance measures, namely: L_1, L_2, shortest path, and Levenshtein.
—We have included a proof of correctness for the `PBA2` algorithm. Essentially, we provide proofs of some important lemmas that guarantee the correctness of the exact score computation process applied by `PBA2`. This new material is given in Section 4.4.1.
—Algorithms `PBA1` and `PBA2` are based on several pruning rules. The effectiveness of pruning is a significant performance factor. In Section 5.3, we give a short discussion on this issue, whereas Figure 14 depicts the number of eliminated objects for the CAL dataset.
—In the performance evaluation, we have included a study of the main memory footprint required by the priority queue data structure. It is shown that the additional memory requirements of the algorithms are very small.
—Finally, we provide in Section 6.4 a preliminary study of an approximate solution. In particular, we show that if we group query objects, it is possible to achieve better performance by penalizing the accuracy of the results.

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` and `ABA`, which are based on adaptations performed on existing algorithms. Then,

in Section 5, we provide the details for the proposed techniques PBA1 and PBA2 as well as the theoretical infrastructure that these algorithms are based on. Section 6 offers performance evaluation results based on diverse datasets and distance functions. Finally, Section 7 concludes the work and discusses briefly future work in the area.

2. RELATED WORK

In this section, we discuss related work, focusing on query types and concepts related to our research, in order to keep the article self-contained. In particular, we cover briefly (1) top-k queries, (2) skyline queries, (3) top-k dominating queries, and (4) nearest-neighbor queries. Top-k queries are at the core of our research, since our proposed algorithms extend existing algorithms that have been proposed previously for general top-k processing. The concept of the skyline is used by one of the baseline algorithms that we study in Section 4. Finally, nearest-neighbor queries are one of the core components to facilitate progressive computation.

2.1. Top-k Queries

Top-k queries model a fundamental notion of human behavior, whereby objects are ranked according to some criteria (e.g., importance, urgency, etc.), and the k highest-ranked items are then selected. Examples are prevalent across settings, such as "top-10 movies of all times," "top-5 best guitarists," "top-20 places to visit," and so on.

As a result of the query's ubiquity, there is a large body of work focusing on efficient algorithms for top-k query processing. Top-k queries have been successfully applied in both structured (e.g., relational databases) and unstructured domains (e.g., document collections) [Dwork et al. 2001]. For example, web search engines are based on top-k query processing to return the most similar documents with respect to a user's query [Culpepper et al. 2012]. In addition, top-k queries are an excellent alternative processing mechanism in database systems [Ilyas et al. 2008] as well as in web-accessible data stores [Marian et al. 2004]. The challenge is to provide efficient algorithms in order to deliver the k best objects by avoiding the score computation of every single object in the collection.

Top-k queries are strongly related to the *ranking* (or *scoring*) *function* used to compute the score of the objects. The score quantifies the quality of an object, allowing us to order the query set. Fagin has significant contributions in the area [Fagin et al. 2001], proposing several algorithms such as TA (Threshold Algorithm) and NRA (No Random Access). These algorithms require attribute values to be presented in descending (or ascending) order, to enable the effective use of thresholding.

Based on the pioneering work of Fagin, several novel techniques have been reported in the literature for top-k query processing. In addition to top-k selection queries based on monotone ranking functions, more complex operators such as *joins* have been studied thoroughly [Ilyas et al. 2004].

2.2. Skyline Queries

Skyline queries, in the context of databases, were initially proposed in Börzsönyi et al. [2001], and since then, they have attracted considerable attention by the database and data analysis community. This surge of interest is due to the query's simplicity and expressive power: they support multiobjective optimization without the need for user-defined scoring functions. The only input required by the user is the *preferences* regarding the minimization/maximization of attribute values. For example, if *price* and *quality* are two of the attributes, then users prefer to minimize price and maximize quality by selecting items that are (objectively) better than (i.e., *dominate*) others.

Several algorithms have been presented to compute the skyline of a dataset, with BBS [Papadias et al. 2005a] being the most preferred when using an R-tree index, due to its progressiveness and I/O optimality. Efficient algorithms have also been proposed in Das Sarma et al. [2009] and Sheng and Tao [2011] for such cases where indexing cannot be applied.

Unlike top-k queries, skyline queries cannot control the size of the output. In fact, depending on the data distribution and dimensionality, it is very likely that the skyline will contain a significantly high number of points. More formally, the expected skyline cardinality of a set of n randomly generated points in d dimensions is $m = O((\ln n)^{d-1})$ [Bentley et al. 1978]. In a dataset containing 10^9 points, having about 10^3 skyline points may not be that much compared to the dataset cardinality, but it is impractical for the user to inspect manually. This problem has long been a major criticism against skyline queries and is what top-k dominating queries—which we focus on in our current work—are trying to address.

2.3. Top-k Dominating Queries

Top-k dominating queries were first introduced in Papadias et al. [2005a] as an alternative to the other widely used preference-based queries, such as general top-k and skyline queries, which we described in the previous paragraphs. A more detailed study of this query type was later performed in Yiu and Mamoulis [2007, 2009].

The main advantages of top-k dominating queries are as follows: (1) the ranking provided is quite intuitive, (2) it does not require specialized scoring functions (as opposed to top-k queries), and (3) the size of the output is controlled by the parameter k (in contrast with skyline queries). Among the algorithms proposed in the literature, CBT [Yiu and Mamoulis 2007] shows the best overall performance. However, it requires the existence of an aggregate R-tree index [Lazaridis and Mehrotra 2001] 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 a small subset of the available attributes.

In Lian and Chen [2009], an algorithm has been proposed supporting top-k dominating queries in uncertain databases. The proposed approach shares the same limitations with CBT, since it is again based on aggregate R-trees. The novelty of this method is that it handles uncertainty in a clear and meaningful way in applications where uncertainty cannot be avoided (e.g., GPS locations). In Zhang et al. [2010], a randomized algorithm is proposed that supports 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 are low. The concepts of top-k dominating queries have been also used in Skoutas et al. [2009]. That work studies web service discovery issues by using dominance relationships through multicriteria matching. Finally, continuous monitoring of top-k dominating query results has been studied in Kontaki et al. [2012] by taking a sliding-window approach.

The common denominator among 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 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 [Sharifzadeh and Shahabi 2006] and metric spaces [Chen and Lian 2008, 2009; Fuhry et al. 2009; Deng et al. 2007], 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 distance function. This seems a natural generalization, taking into account that many

modern applications rely solely on triangular inequality to support similarity queries [Chávez et al. 2001].

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 a significant and desirable 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) [Papadias et al. 2005a], which has been designed to answer skyline queries in multidimensional datasets indexed by an R-tree. Therefore, BBS is not applicable in our case. Moreover, although the algorithm in Chen and Lian [2009] works with metric-based access methods, it is designed to report the skyline, and it is not equipped to handle the concept of the dominance score.

Tiakas et al. [2011] study efficient progressive algorithms for top-k dominating queries in multidimensional datasets, where the user may select a subset of the available dimensions. These techniques assume a vertical decomposition of the dataset, where each dimension is organized separately. Unfortunately, this requirement renders them unfit for the metric case explored in this article. However, some results from Tiakas et al. [2011] are adapted and utilized in our current work, in order to provide (1) efficient score computation and (2) effective pruning.

2.4. Nearest-Neighbor Processing

From the brief outline of our top-k dominating technique for metric spaces presented in Section 1, one can easily see that *incremental nearest-neighbor* processing is a core component of our approach. Given its integral role, we provide a short survey of techniques and results in this area.

Determining the k nearest neighbors of a query object is a fundamental operation, due to its importance in database query processing as well as in machine-learning algorithms (e.g., k-NN classification). Nearest-neighbor queries have been successfully applied in a broad spectrum of domains, such as spatial databases [Roussopoulos et al. 1995], spatiotemporal databases [Raptopoulou et al. 2003], sensor networks [Xu et al. 2007], and metric spaces [Ciaccia et al. 1997], to name a few. Typically, given a query object q and an integer k, the output of a k-NN query contains the k closest objects with respect to q, where proximity is computed by means of a distance function. Note that nearest-neighbor queries can be considered as a special case of top-k queries, where the ranking function corresponds to the distance among the objects.

In this work, we are mostly interested in incremental nearest-neighbor computation, where given a query object q, we require by the access method to deliver the *next best object* upon request. This means that we do not provide the number of nearest neighbors (k) in advance, and each neighbor is delivered by executing a get-next operation incrementally. The concept of incremental nearest-neighbor computation has been introduced in Hjaltason and Samet [1995] in spatial databases.

The requirement for incremental retrieval allows us to support progressiveness of result output. In this work, we are using incremental nearest-neighbor queries over metric access methods. Since the number of active incremental nearest-neighbor queries equals the number of query objects, the efficient computation of the final result is not straightforward.

3. FUNDAMENTAL 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 I depicts the basic notations that are frequently used in the upcoming sections.

Table I. Frequently Used Symbols

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 objects
$d()$	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
$dom(p) = dom(p, Q)$	exact dominance score of p, with respect to Q
$estdom(p) = estdom(p, Q)$	estimated dominance score of p, with respect to Q
$eq(p)$	number of objects equivalent to p
$MSS(Q)$	metric space skyline with respect to Q
$MSD(Q, k)$	top-k dominating objects with respect to Q

3.1. Basic Definitions

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

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

Then, d is called a *metric function* and the pair $(\mathcal{U}, d)$ is called a *metric space*. If the objects of the metric space are tuples (records) with numeric attributes, then the pair $(\mathcal{U}, d)$ 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. Practically, we are interested in a subset of D of $\mathcal{U}$, which is called the *dataset* and contains the underlying data objects of interest.

Definition 3.1. Let D be a dataset, $q \in D$, 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 3.2. Let $a_1, \ldots, a_m$ be the m attributes defining the objects. A ranking function f is monotone if $f(a_1, \ldots, a_m) \leq f(a'_1, \ldots, a'_m)$ whenever $a_i \leq a'_i, \forall i$.

Definition 3.3. If (D, d) is a metric space, Q is a set of query objects $Q = \{q_1, q_2, \ldots, q_m\}$, and $f()$ is a monotonically increasing function, then the *aggregate distance* between an object p and the query set Q is defined as follows:

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

The result of an *aggregate nearest -neighbor query*, denoted as $ANN(Q, k)$, contains the k objects with the minimum aggregate distance computed according to 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 Papadias et al. [2005b]).

Definition 3.4. Let (D, d) be a metric space and Q be a set of query objects $Q = \{q_1, q_2, \ldots, q_m\}$. 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 strictly smaller distance than r to at least one query object. The set of objects in D that 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 Yiu and Mamoulis [2007], top-k dominating queries were defined for objects with fixed coordinates. In this article, 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, that is, $dom(p) = |\{r \in D : p \prec r\}|$. $dom(p)$ is referred to as the *dominance score* of p. 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 3.5. Two objects p_1 and p_2 are called *equivalent* with respect to a query set $Q = \{q_1, q_2, \ldots, q_m\}$ if $\forall i : 1 \leq i \leq m, dist(p_1, q_i) = dist(p_2, q_i)$.

We can also generalize equivalence, denoting by $eq(p)$ the set of points from the dataset D that are equivalent to p, with respect to a query set Q. More formally, $eq(p) = \{x \in D : dist(p, q_i) = dist(x, q_i), q_i \in Q, \forall i, 1 \leq i \leq m\}$.

Definition 3.6. Let $Q = \{q_1, q_2, \ldots, q_m\}$ be a set of query objects, and consider a k-nearest-neighbor query $NN(q_i, k)$ for each query object q_i. An object $p \in D$ is considered a *common neighbor* if and only if it is found among the k-nearest neighbors for every $NN(q_i, k)$. Formally, p is a common neighbor if $p \in \bigcap NN(q_i, k)$, $1 \leq i \leq m$.

In the upcoming sections, 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 factor, dominating the overall processing cost. Note that this problem poses some nontrivial challenges such as the following: (1) attributes and their values are generated dynamically, depending on the query set Q; (2) the dominance relationship can only be instantiated once the set of query objects Q is provided; and (3) progressive computation is desirable.

Definition 3.7. Let $q_j \in Q$ be a query object and $o_i \in D$ be an object from the dataset. Then, the *rank position* of o_i with respect to q_j is k, denoted by $rank(q_j, o_i) = k$, if o_i is the kth nearest neighbor of q_j.

We can also use the alternative notation $q_{jk} = o_i$, which has the same exact meaning: the kth nearest neighbor of q_j is o_i. More generally, we denote $q_{j1}, q_{j2}, \ldots, q_{jn}$ the nearest-neighbor order of the n objects from query object q_j, $\forall j = 1, \ldots, m$. As an example, q_{j2} is the second-nearest neighbor with respect to query object q_j. This notation allows us to center our focus on the query objects and present its ranking without knowing which item $o_i \in D$ is ranked at position k.

The objects are ordered in ascending order of their distance from q_j, that is, $d(q_j, q_{j1}) \leq d(q_j, q_{j2}) \leq \cdots \leq d(q_j, q_{jn}), \forall j = 1, \ldots, m$. Also, note that the n objects are not necessarily ordered in the same way across query objects. Therefore, an object $o_k \in D$ can be ranked second for q_i, while it may be ranked 30th for q_j.

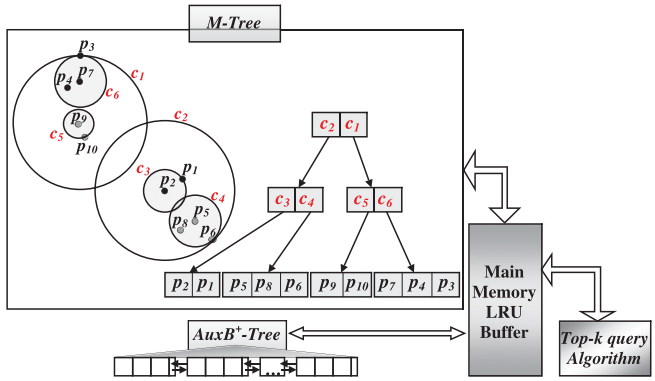

Fig. 3. Access methods employed.

3.2. Indexing Structures for Metric Spaces

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 should be employed toward efficient processing. Among the available metric-based access methods studied in the literature [Bozkaya and Ozsoyoglu 1999; Brin 1995; Chávez et al. 2001; Hjaltason and Samet 2003], we select the M-tree [Ciaccia et al. 1997], which is well appreciated due to its simplicity, its resemblance to the B-tree, its excellent performance, and its ability to handle dynamic datasets (i.e., insertions and deletions). However, our methods are orthogonal to the indexing scheme used, as long as incremental nearest-neighbor queries are supported.

Figure 3 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 was retrieved during scanning, a counter that is used for exact score computation during backward scanning, and its current max-rank position in the nearest-neighbor order from the query objects. During execution, the scanned object IDs 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 the I/O cost due to locality of references and thus helps in reducing the total computation time.

3.3. Fagin's Threshold Algorithm

As stated previously, our proposal uses some concepts present in Fagin's Threshold Algorithm studied in Fagin et al. [2001]. Here, we discuss briefly the main concepts of TA in order to keep the article self-contained. To illustrate the way TA works, we assume that the dataset is composed of n tuples, where each tuple contains m numeric attributes. Therefore, each object can be considered as a point in the mth dimensional space. We will also assume that the scoring function is the `sum` of the attribute values. TA reports the k-best objects with respect to the scoring function used. Instead of using `sum`, other monotone scoring functions can be used equally well.

For simplicity, assume that each attribute is sorted in nondecreasing order of the attribute values. This is equivalent to a table containing n rows (the number of objects) and m columns (the number of attributes), where each column is stored in nondecreasing order of its values. TA works in two alternating phases: sorted access and random

access. A sorted access corresponds to fetching the objects' IDs and the associated attribute values from the next row of the table. Evidently, in the first sorted access the first row is retrieved, in the second sorted access the second row is retrieved, and so on. Let p denote the position of the last row retrieved by a sorted access. The values contained in this row in general correspond to different objects. If all attribute values correspond to the same object (which is a special case), then we could immediately determine the score of this object. Otherwise, a sequence of random accesses takes place in order to determine the missing values.

TA maintains a buffer of the best k objects determined so far. Therefore, every time the score of a newly discovered object is determined, we must check if this object should be inserted in the buffer. If not, the object is discarded. In order to decide if we should continue with more sorted accesses, a threshold T is determined that equals the sum of the attribute values of the current row (p). If T is larger than all scores currently stored in the buffer, then the algorithm terminates and reports the k objects determined so far. Otherwise, the algorithm proceeds with the next sorted access, fetching the next row in position $p + 1$.

4. ADAPTING EXISTING ALGORITHMS

As a starting point for processing the top-k dominating queries, we provide algorithmic solutions that build on existing techniques and that we adapt accordingly. We present two such algorithms: (1) the Skyline-Based Algorithm (SBA) and (2) the Aggregation-Based Algorithm (ABA). These two algorithms use different methodologies to provide the result. In particular, SBA exploits skylines, whereas ABA uses aggregation. Therefore, if either of these two functionalities is present, metric-based top-k dominating queries can be supported. Both algorithms assume the existence of an M-tree as the primary access method. However, any metric-based access method can be applied equally well.

4.1. The Skyline-Based Algorithm

The first algorithm we study is based directly on the observation of Papadias et al. [2005a] that the top-1 object of a monotone scoring function belongs to the skyline. As the following lemma states, this fact is valid in the case of the top-1 dominating object as well.

LEMMA 4.1. *The top-1 dominating object is always a metric skyline object, that is,* $MSD(Q, 1) \subseteq MSS(Q)$.

PROOF. The top-1 dominating object t has the maximum dominance score in D, that is, $dom(t, Q) \geq dom(r, Q), \forall r \in D$. 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: compute the skyline S of D, 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 Fuhry et al. [2009], which is the state-of-the-art algorithm for general metric-based skyline queries, since it outperforms previously proposed algorithms, such as MSQ [Chen and Lian 2008, 2009]. 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 in the $AuxB^+$-tree. Therefore, both random and sorted accesses are supported.

ALGORITHM 1: SBA(D,Q,k)

Input: D dataset, 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) + +;
09.       if (dom(o) > Max)
10.          Max ← dom(o);
11.          TopObject ← o;
12.    report TopObject;
13.    remove TopObject from D;
```

SBA reports the top-k results in a progressive manner. However, this method has two important limitations: (1) it performs many unnecessary score computations, since the skyline is often larger than k, and (2) when there is a large number of query objects ($|Q|$), the skyline grows significantly and in some cases approaches the dataset cardinality $|D|$. These characteristics may lead to significant performance degradation due to increased processing costs.

4.2. The Aggregation-Based Algorithm

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

LEMMA 4.2. *If an object p dominates another object r, then p will be returned before or together with r in the result list of a top-h aggregate nearest-neighbor query (ANN(Q, h)) by using any monotonically increasing aggregate distance function $f(.)$. Formally: $p \prec r \Rightarrow adist(p, Q) \leq adist(r, Q)$, where $adist(p, Q) = f(d(p, q_1), d(p, q_2), \ldots, d(p, q_m))$ and $adist(r, Q) = f(d(r, q_1), d(r, q_2), \ldots, d(r, q_m))$.*

PROOF. Let p and r be two objects such that $p \prec r$. From the definition of dominance in metric spaces, it holds that

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

Due to the monotonicity of $f(.)$, it holds that

$$\forall q_i \in Q, d(p, q_i) \leq d(r, q_i) \Rightarrow f(d(p, q_1), \ldots, d(p, q_m)) \leq f(d(r, q_1), \ldots, d(r, q_m)) \Leftrightarrow$$

$$\Leftrightarrow adist(p, Q) \leq adist(r, Q).$$

Therefore, for any monotonically increasing aggregate distance function, the object p has a smaller than or equal aggregate distance to r, and it is located before or along with r in the aggregate nearest-neighbor query results. □

Lemma 4.2 associates a top-k dominating query with a top-h aggregate nearest-neighbor query using any monotone aggregate function (e.g., sum, avg, min, max). For the rest of the discussion, we will assume for simplicity that the aggregate function used is the sum. However, in the performance evaluation in Section 6 we provide some results for other aggregation functions. Based on the previous lemma, the result of

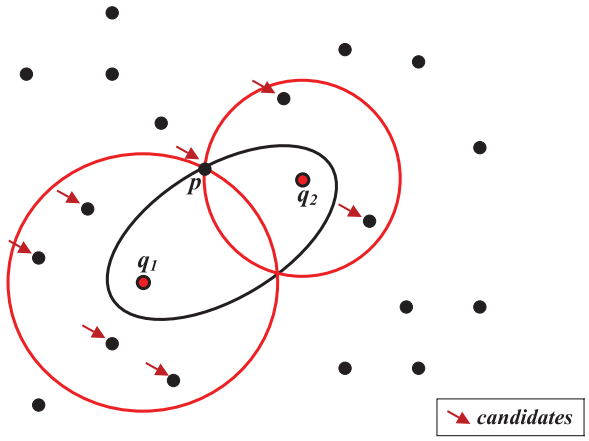

Fig. 4. Example for ABA description.

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 $h \geq k$), that is, $MSD(Q, k) \subseteq ANN(Q, h)$. 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 4.2, Figure 4 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 dataset. 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 Sharifzadeh and Shahabi [2006] 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 in 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 that dominate p have smaller sum-aggregate distances than p.

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 Papadias et al. [2005b] and Sharifzadeh and Shahabi [2006]. The main difficulty, however, is that we do not know in advance a convenient value of h such that $MSD(Q, k) \subseteq ANN(Q, h)$, which will allow us to answer a top-k dominating query. Even if such an h value were known, it would have to be much larger than k to retrieve the correct top-k result, and the performance of such an algorithm would be significantly impacted. Finally, the algorithm would operate under a *filter-and-refine* scheme, where h objects are retrieved first and then we discard objects until we have the top-k result, which we will report all at once at the end of query execution. It is easy to see that such an algorithm would invalidate the progressiveness property, and as such would not meet our requirements.

Therefore, we pursue an alternative methodology, which is based on the following lemma.

LEMMA 4.3. *The first sum-aggregate nearest neighbor of Q is always a metric space skyline object, that is, $ANN(Q, 1) \subseteq MSS(Q)$.*

PROOF. Let t be the top-1 *sum*-aggregate nearest-neighbor object of Q, that is, $t = ANN(Q)$. Then t cannot be dominated by any other object, because if we assume that there is an object x that dominates t, then using Lemma 4.2, we get $adist(x, Q) < adist(t, Q)$, which contradicts that t is the top-1 *sum*-aggregate nearest-neighbor object of Q. Therefore, t is definitely a skyline object. □

Initially, the top-1 sum-aggregate nearest-neighbor object p of Q is computed, that is, $p = ANN(Q, 1)$. Following Lemma 4.3, p is a skyline object and, as a result, there are no objects that dominate p. This means that there are no objects inside p's dominator region and, additionally, p dominates all other objects lying in its dominance region. This result ensures that the top-1 dominating object cannot lie in the dominator/dominance regions of p. Thus, we should search for the top-1 object in the rest of the dataset. Candidates for the top-k result are collected in set C by performing simple range queries centered at the query objects $q_1, q_2, \ldots, q_m$ with radius $d(p, q_1), d(p, q_2), \ldots, d(p, q_m)$, 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.

In order to have a better view of those candidates, let us consider the example of Figure 4. 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.

The outline of ABA is given in Algorithm 2. For the aggregate nearest-neighbor query $ANN(Q)$ of Line 2, we use the MBM algorithm of Papadias et al. [2005b], which is the state-of-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 stored and updated into the $AuxB^+$-tree.

ALGORITHM 2: ABA(D,Q,k)

Input: D dataset, 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;
07.     Max ← 0;
08.     TopObject ← ∅;
09.     for each object o ∈ C;
10.         dom(o) ← 0;
11.         for each object x ∈ D, x ≠ o
12.             if (o ≺ x) dom(o) + +;
13.         if (dom(o) > Max)
14.             Max ← dom(o);
15.             TopObject ← o;
16.     report TopObject;
17.     remove TopObject from D;
```

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: (1) it recalculates up to k times the dominance values of some nearest-neighbor candidate objects of C; (2) 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 (3) when the query objects are far from each other, the range queries may return a large number of candidates, and thus C grows significantly, leading to high computational costs.

5. PROPOSED APPROACH: THE PRUNING-BASED ALGORITHM

As we pointed out in the previous paragraphs, algorithms SBA and ABA are faced with some severe shortcomings, mainly a high degree of recomputations and a possibly high memory footprint, depending on the cardinality of the query set Q. For these reasons, a more efficient alternative is necessary for processing top-k dominating queries in metric spaces. Therefore, in this section, we present and discuss in detail our proposed approach, the Pruning-Based Algorithm (PBA).

The key idea behind the Pruning-Based Algorithm is to incrementally retrieve the nearest neighbors of the query objects in a round-robin fashion and compute the dominance scores of their common neighbors. PBA also employs early termination criteria, allowing us to to extract the top-k results without visiting all of the points in D (unless absolutely necessary), thereby reducing computational costs even further. We briefly introduced these notions in the beginning of our work, with an illustrative example (Figure 2) of a top-3 dominating query in a 2-dimensional Euclidean space.

Our technique comes in two variations, PBA1 and PBA2, both of which operate within the same framework and share the properties of PBA. We will discuss the two variants, and their differences, extensively in the following paragraphs, but first we provide a general overview of their common ground, PBA. In short, PBA incorporates the following ideas, which contribute significantly to the reduction of CPU time, I/O cost, and the number of distance computations performed:

—Efficient techniques are used to compute the score of an object, which are based on incremental nearest-neighbor search combined with a round-robin examination, resulting in more efficient processing.
—Effective pruning rules are employed, toward eliminating as early as possible objects that cannot be part of the answer.

5.1. Theoretical Foundations of PBA

PBA's correctness relies on a set of properties that associate the dominance relation with the rankings of the objects, while these are retrieved during the incremental nearest-neighbor search. These properties, which we theoretically prove in the following paragraphs, also serve as the basis for efficient processing of top-k dominating queries in metric spaces. They allow us to estimate upper bounds of the dominance score of an object from D and can be used to apply effective pruning, thereby reducing computational and memory costs of our proposed approach.

We begin with those properties related to the dominance relation alone and proceed with theoretical results that focus on the dominating power of an object $o \in D$.

LEMMA 5.1. *When we have more than one query object $q_1, \ldots, q_m$ ($m = |Q| > 1$) 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 $p_{j1}, p_{j2}, \ldots, p_{jn}$ be the nearest-neighbor order of the n objects from the query object q_j, $\forall j = 1, \ldots, m$. First, we examine the case that these orderings are unique (without equal distances from the query objects), that is,

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

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

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

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

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 5.2. *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 that 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 in 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 Ciaccia et al. [1997]. Incremental retrieval is performed by using all query objects in a round-robin fashion (i.e., first *NN* from $q_1, \ldots,$ first *NN* from q_m, second *NN* from $q_1, \ldots,$ second *NN* from q_m, etc.). This idea was first proposed in the *Threshold Algorithm* of Fagin et al. [2001] and has been subsequently employed in several other problems, such as distributed skyline queries [Balke et al. 2004] and aggregate nearest-neighbor queries [Papadias et al. 2005b].

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 along with its estimated score. Therefore, the heap will always contain 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 5.3. *If o_a and o_b are the top-2 common neighbor objects in 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. □

Lemma 5.3 is the reason for the progressive behavior of PBA. If the common neighbor object o_a, currently under examination, satisfies the condition of Lemma 5.3, 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.

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 as shown in 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 IDs of the retrieved common neighbor objects determined so far along with their corresponding dominance values (exact or estimated). This reduces the memory footprint of the employed technique while allowing for the efficient retrieval of supplementary information.

We also note that to compute $eq(o)$, we continue the round-robin scan until we read all the objects of the equality group of o for query object q_j that has been retrieved last (i.e., until we find an object x in the nearest-neighbor order of q_j such that $d(x, q_j) > d(o, q_j)$).

ALGORITHM 3: PBA(D,Q,k)

Input: D dataset, 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
```

PROCEDURE 1: NEXTCOMMONNEIGHBOR(D, Q, H)

00. $j = 001$. **do**
02. j++; if (j=m+1) then j=1; (i.e., the next query object q_j is selected)
03. $o \leftarrow$ NEXTNEARESTNEIGHBOR (q_j);
04. **insert** o in $AuxB^+$ (if not present) and increase $q_{counter}$ by 1;
05. **while** $q_{counter}$ of o is less than m ($m = |Q|$);
06. compute number of equivalent objects of o, $eq(o)$ and insert them in H;
07. $estdom(o) \leftarrow n - \max_j rank(o, q_j) + eq(o)$;
08. $H.enheap(o)$;

5.2. Reducing the Cost of Score Computation

The computation of the exact dominance score of an object (Procedure EXACTSCORE of Line 11 in Algorithm 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 that is based on *reverse scanning* [Tiakas et al. 2011]. Note that the algorithms studied in Tiakas et al. [2011] work with a multidimensional space, and therefore reverse scanning must be adapted in our case to work with a metric space. We denote this type of computation by EXACTSCORE-RS and the algorithm that uses it 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 $q_j, \forall j = 1, \ldots, m$: $U = \bigcup_{j=1}^{m} U_j$. The following guarantees the computation of the exact score of o:

LEMMA 5.4. *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. First, o cannot dominate the following objects: (1) any object $x \in U$, as it holds $d(q_j, x) < d(q_j, o)$ for some q_j; (2) its equivalent objects; and (3) itself.

Moreover, for any other object y, it holds that y is different from o, it is not equivalent to o, and it is not in U. Therefore, since $y \notin U$, it holds that $d(q_j, y) \geq d(q_j, o), \forall j = 1, \ldots, m$, and as y is not equivalent with o, it holds that $\exists j \in \{1, \ldots, m\} : d(q_j, y) > d(q_j, o)$. Thus, y is definitely dominated by o. □

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, the required counting can be successfully performed inside the $AuxB^+$-tree, without materializing the sets U_j and U. Moreover, the formula of Lemma 5.4 requires only the cardinality of the set U and not its contents. The outline of the score computation method using the reverse-scanning technique is given in Procedure 2. For the algorithm description, aux is the number of all unique objects inserted into the $AuxB^+$-tree to the time that this procedure is called, that is, the current $AuxB^+$-tree size (this information is kept in its root and can be retrieved immediately). Moreover, u is an integer variable that keeps and updates the cardinality of U. However, during the process of this counting, we must change some of the values of the $q_{counter}$ counters for some objects that are already inserted in the $AuxB^+$-tree, but we must keep intact the current values of the $q_{counter}$ counters as they must be used for the detection of the next common neighbor later. Therefore, inside the record of an object x in $AuxB^+$-tree, we use a copy

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. **if** ($d(q_j, x) \geq d(q_j, o)$)**then** decrease $qc_{counter}$ of x in $AuxB^+$-tree by 1;
06. **if** ($qc_{counter}$ of $x = 0$) **then** $u = u - 1$;
07. **while** $d(q_j, x) \geq d(q_j, o)$
08. **end-for**
09. $dom(o) = n - u - eq(o) - 1$;
10. **return** $dom(o)$;

of its $q_{counter}$, the $qc_{counter}$, and we update only the $qc_{counter}$ values. Let us describe this procedure in more detail:

When the EXACTSCORE-RS procedure is called, object o has already been retrieved from all query objects (it is a common neighbor object). 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 orders from the query objects may have been counted (have $q_{counter} > 0$), but they are not included in U. In order to count the objects of U correctly, we scan backward from each query object q_j (Lines 3 and 4) in the corresponding nearest-neighbor orders until we find an object x such that $d(q_j, x) < d(q_j, o)$ (Line 7). During the reverse-scanning process, for every object x that has $d(q_j, x) \geq d(q_j, o)$ (Line 5), we decrease its $qc_{counter}$ in $AuxB^+$-tree by 1 (Line 5). Every time the $qc_{counter}$ of an object x becomes zero (Line 6), that object must be excluded from the union set U (because that means that it does not appear at all before o in the nearest-neighbor orders), and thus we decrease u by 1.

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).

The exact score computation can also be 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, that is, $Lpos_{o_i}(q_j) = \min_h\{rank(o_h, q_j) : d(q_j, o_h) = d(q_j, o_i)\}$. It is important to note that the $Lpos$ rank positions are recorded in the $AuxB^+$-tree during the incremental nearest-neighbor retrieval together with the q counters, and thus they are available before the computation of 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 position values already recorded in the $AuxB^+$-tree, while for the second group only some of their $Lpos$ rank positions values have been recorded. The reason is that objects from the second group have been detected by only some query objects q_j. By analyzing these cases, the following lemmas guarantee the calculation of the exact score of o.

In order to get a better insight of the proofs, consider Figure 5, which presents a snapshot of the PBA algorithmic procedure. In particular, Figure 5 depicts the nearest-neighbor orders from the query objects and the current detected common neighbor object o together with its equal-distance object groups. We denote by pos the maximum rank position retrieved so far, and 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).

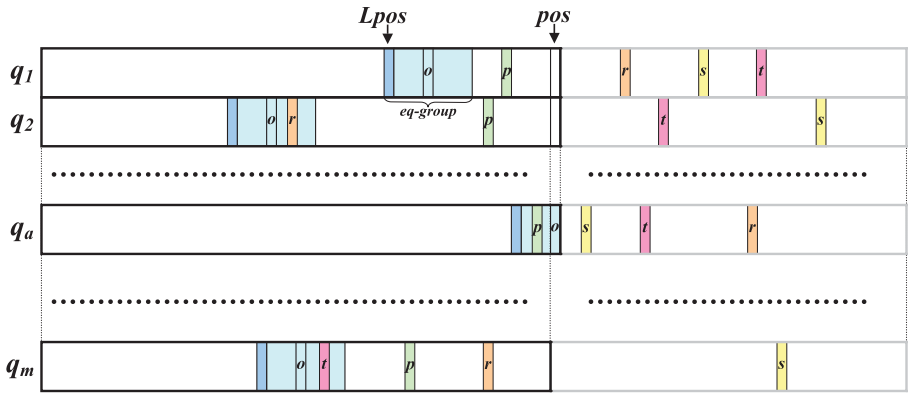

Fig. 5. Example used in the proof of correctness for PBA2 algorithm.

LEMMA 5.5. *For any object o_i that has been inserted into the AuxB$^+$-Tree, if it holds that $Lpos_{o_i}(q_j) \geq Lpos_o(q_j)$ for all the available Lpos values of o_i, and there exists at least one q_j such that $Lpos_{o_i}(q_j) > Lpos_o(q_j)$, then o dominates o_i.*

PROOF. If all *Lpos* values of o_i are available, then the condition $Lpos_{o_i}(q_j) \geq Lpos_o(q_j)$ ensures that $d(q_j, o) \leq d(q_j, o_i), \forall j = 1, \ldots, m$. Moreover, the condition $Lpos_{o_i}(q_j) > Lpos_o(q_j)$ for some q_j ensures that $\exists j : d(q_j, o) < d(q_j, o_i)$ holds and o definitely dominates o_i. In Figure 5, object p is a representative of that case. Observe that all the objects of that case must lie after or inside the equal-distance object groups of o.

In case not all *Lpos* values of o_i are available but only some of them are, then the condition $Lpos_{o_i}(q_j) \geq Lpos_o(q_j)$ ensures that $d(q_j, o) \leq d(q_j, o_i)$ only for some values of j (corresponding to the available *Lpos* values). Moreover, the condition $Lpos_{o_i}(q_j) > Lpos_o(q_j)$ for some q_j ensures that $\exists j : d(q_j, o) < d(q_j, o_i)$ for one of those values of j. For the remaining values of j, the object o_i has not been detected from the corresponding query objects q_j, in which case either o_i lies in a next equal-distance object group with $Lpos_{o_i}(q_j) > Lpos_o(q_j)$ or it lies in the same equal-distance object group with o (provided that this group has not been completely discovered yet) having $Lpos_{o_i}(q_j) = Lpos_o(q_j)$. Therefore, in either case, for all the remaining values of j, it holds that $d(q_j, o) \leq d(q_j, o_i)$ and o definitely dominates o_i. In Figure 5, object r is a characteristic example of that case. Observe that r has been detected by some query objects, but not by all of them. For those query objects q_j by which o has not been encountered already, it definitely lies after the position *pos*. □

LEMMA 5.6. *Any object o_i that has not been inserted yet into the AuxB$^+$-tree is dominated by or is equivalent to the current detected common neighbor object o.*

PROOF. As o_i has not been inserted yet into the AuxB$^+$-tree, none of its *Lpos* values are available. On the contrary, all *Lpos* values of the current detected common neighbor object o are available. Therefore, for all values of j, the object o_i has not been detected from the corresponding query objects q_j. Consequently, it either lies in a next equal-distance object group with $Lpos_{o_i}(q_j) > Lpos_o(q_j)$ or it lies in the same equal-distance object group with o having $Lpos_{o_i}(q_j) = Lpos_o(q_j)$, but the group has not been fully discovered yet. Whatever the case may be, it holds that $d(q_j, o) \leq d(q_j, o_i), \forall j = 1, \ldots, m$, meaning that o_i is either dominated by or equivalent with o. In Figure 5, the object s is a representative of that case. Observe that s has not been detected yet by a query object q_j, placing it after the position *pos* for all j values. □

LEMMA 5.7. *For any object o_i that has been inserted into the AuxB^+-Tree, if it holds that $Lpos_{o_i}(q_j) = Lpos_o(q_j)$ for all the available Lpos values of o_i, then o dominates o_i or the objects o and o_i are equivalent.*

PROOF. If all *Lpos* values of o_i are available (i.e., it has been detected by all query objects during the incremental nearest-neighbor retrieval), then the condition $Lpos_{o_i}(q_j) = Lpos_o(q_j)$ ensures that $d(q_j, o) = d(q_j, o_i), \forall j = 1, \ldots, m$. As a result, the two objects o and o_i are definitely equivalent.

In case some *Lpos* values of o_i are missing, while the rest of them are already available, then the condition $Lpos_{o_i}(q_j) = Lpos_o(q_j)$ ensures that $d(q_j, o) = d(q_j, o_i)$ holds for some values of j, corresponding to the available *Lpos* values. For the remaining values of j, the object o_i has not been detected from the respective query objects q_j. Therefore, either o_i lies in a next equal-distance object group, having $Lpos_{o_i}(q_j) > Lpos_o(q_j)$, or it lies in the same equal-distance object group with o, but the group has not been completely discovered yet. In that latter case, it holds that $Lpos_{o_i}(q_j) = Lpos_o(q_j)$. In either case, for all values of $j = 1, \ldots, m$, it holds that $d(q_j, o) \leq d(q_j, o_i)$, meaning that either o dominates o_i or the objects o and o_i are equivalent. In Figure 5, this case is demonstrated by object t, which still remains undetected by some query objects q_j. Consequently, for those j values, object t lies after the current position *pos* and is dominated by the current common neighbor o. □

Lemmas 5.5, 5.6, and 5.7 cover all of the alternatives whereby an object o can dominate another object o_i. In any other case, there will be at least one value of j for which $Lpos_{o_i}(q_j) < Lpos_o(q_j) \Rightarrow d(q_j, o) > d(q_j, o_i)$. For these particular cases, o cannot dominate o_i. Briefly explained, the new alternative for the exact score computation works as follows: (1) it performs one sequential scan over the current contents in the leaves of the *AuxB^+*-Tree, (2) it checks the appropriate conditions for the *Lpos* values, and (3) it counts the dominations of the current detected common neighbor object o. The variant of PBA that uses this new technique for the exact score computation (called EXACTSCORE-AUX and outlined in Procedure 3) is denoted as PBA2. Therefore, the two variations, PBA1 and PBA2, differ only in the way exact scores are computed.

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 ∈ AuxB+-tree
04.   ff = true
05.   for each query object q_j ∈ Q
06.     if (Lpos_oi(q_j) ≠ NULL) ∧ (Lpos_oi(q_j) < Lpos_o(q_j))
07.       ff ← false; break;
08.   end-for
09.   if (ff = true)
10.     fe ← true
11.     for each query object q_j ∈ Q
12.       if (Lpos_oi(q_j) ≠ Lpos_o(q_j))
13.         fe ← false; break;
14.     end-for
15.     if (fe = true) ff ← false
16.   end-if
17.   if (ff = true) dom_in + +;
18. end-for
19. dom(o) ← dom_in + n − |AUX|;
20. return dom(o);
```

5.3. Boosting PBA's Efficiency with Pruning

Algorithms PBA1 and PBA2 enable the usage of several pruning rules, which reduce the runtime costs further. Some of the proposed pruning rules use a global pruning value G for the dominance scores. G is initialized to 0 and 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, that is, $G = dom(o_k) - 1$. Any object with an exact dominance value less than or equal to G can be pruned safely.

We propose three different types of pruning rules:

—A *Discard Pruning Rule* (DPR), allowing us to discard objects that have not yet been retrieved
—An *Early Pruning Rule* (EPR), allowing us to prune objects before the computation of their exact dominance score
—An *Internal Pruning Rule* (IPR), allowing us to prune objects during the process of exact dominance score computation

We note that the previous descriptions correspond to a *family* of rules, considering that each type may be instantiated in various ways. In the following paragraphs, we describe the rules that we have employed in our study. More specifically, we applied a total of seven pruning rules, broken down as follows: one discard rule (DPR), five early-pruning rules ($EPR1$–$EPR5$), and one internal pruning rule (IPR). Each one of these rules corresponds to a different proposition. The proofs are easily derived and are based on results that have been proven previously in the article.

Discard Pruning Rule ($DPR1$): The first time that the exact dominance scores of k objects in the heap are computed, all other objects that have not yet been inserted into the $AuxB^+$-tree are ignored. This rule is based on Lemma 5.1, according to which items that have not been yet encountered will be dominated by items in the top-k result. Objects are inserted into the $AuxB^+$-tree only after they have been retrieved at least once, meaning that any object not in that structure has never been previously seen. Therefore, it is dominated by the k objects with exact scores, which we have in the top-k heap, and can be safely ignored.

Early Pruning Rule 1 ($EPR1$): Let o_k be the object with the kth exact dominating score computed so far. Moreover, let o be the current retrieved common neighbor object for calculations. If $o_k \prec o$, then o can be pruned.

Early Pruning Rule 2 ($EPR2$): Let o_i be any exact dominating object, calculated after the exact top-k dominating object into the heap H. Also, let o be the current retrieved common neighbor object for calculations. If $o_i \prec o$, then o can be pruned.

Early Pruning Rule 3 ($EPR3$): Let o be the current retrieved common neighbor object for calculations. Then, object o can be safely pruned if the following holds:

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

According to Lemma 5.2, object o will be able to dominate at most $n - \max_j Lpos_o(q_j)$ objects. However, this dominance score will be less than or equal to G, which is the dominance score of the current kth best object. As such, object o will not have a better score than the current top-k result, and it can be safely pruned.

Early Pruning Rule 4 (*EPR4*): Let o be the current retrieved common neighbor object for calculations. Then, object o can be safely pruned if the following holds:

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

Early Pruning Rule 5 (*EPR5*): Let o_i be any exact calculated dominating object, present in the heap H. Also, let o be the current retrieved common neighbor object for calculations. Then, o can be pruned if the following holds:

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

Internal Pruning Rule (*IPR1*): Let o be the current retrieved common neighbor object, associated with the following values: $curDom(o)$ is the object's current recorded score, whereas $curPos(q_j)$ is its current rank position in the nearest-neighbor order from q_j during the reverse-scanning process. If during the calculations the following inequality holds:

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

then o can be safely pruned, and the remaining calculations can be skipped.

Early and internal pruning rules are applied to heap objects, whereas the discard pruning rule is applied to both heap and $AuxB^+$-tree elements. Through these rules, we can eliminate a significant number of objects, which contributes greatly toward more efficient processing. In addition, pruning also minimizes the memory footprint of PBA, because it reduces the number of objects stored in the heap structure, residing in main memory. This result is also verified empirically, in our experimental evaluation.

The rules described previously do not all have the same pruning power. Some rules may prune more points, whereas others may prune fewer. Moreover, some rules may start pruning away during the early stages of execution, whereas other rules may take effect after a while. This is not to say that rules of the latter categories are useless. It does tell us, however, that the order in which we apply the pruning rules can significantly impact the performance of the algorithm, with (potentially) noticeable differences in execution runtime. These runtime differences are easily explained if we consider that the longer an object survives (i.e., is not pruned), the more computations will be performed on it. These computations are due to attempts to try to add the object to the top-k result, as well as attempts to try to prune it in later stages of the algorithm.

The pruning rules are applied in a specific order. Rule DPR1 is applied first, as it must be applied at exactly the time that the exact dominance scores of k objects in the heap are computed. Then, EPR rules are applied in an order that is strongly related to their execution cost. More specifically, any EPR rule can be applied immediately after the time that the exact dominance scores of k objects in the heap have been computed, and before the computation of the exact dominance score of the current retrieved common neighbor object o. However, in order to have an efficient plan, EPR rules are applied in the following order:

(i) Rules EPR4 and EPR3 are applied first as they require the retrieval of only the *Lpos* positions of the current examined common neighbor object o from the $AuxB^+$-tree.

(ii) Rule EPR1 follows next, as it requires the retrieval of the *Lpos* positions of the current examined common neighbor object o and of the current exact top-k dominating object in the heap (o_k) from the $AuxB^+$-tree, in order to perform the domination check.
(iii) Rule EPR5 follows next, as it requires the retrieval of the *Lpos* positions of the current examined common neighbor object o and the retrieval of an exact calculated dominating object o_i that is located in the heap, from the $AuxB^+$-tree, in order to perform the domination check. This rule has an additional cost as o_i is selected according to the minimum absolute difference of positions preservation; that is, we select the heap object that minimizes the absolute difference of the *Lpos* positions between the common neighbor objects that are retrieved.
(iv) Rule EPR2 is enforced next, since it requires the retrieval of the *Lpos* positions of the current examined common neighbor object o and the retrieval of all exact calculated dominating objects o_i that have been inserted into the heap after the exact top-k dominating object, from the $AuxB^+$-tree, in order to perform the domination checks.

Finally, rule IPR1 is applied since it requires that the computation of the exact dominance score of the current retrieved common neighbor object o has been already started. Based on the previous discussion, we conclude that the rule execution order in increasing execution cost order is DPR1, EPR4, EPR3, EPR1, EPR5, EPR2, IPR1.

6. PERFORMANCE EVALUATION

In this section, we present a set of representative experimental results, demonstrating the comparative performance of the studied algorithms `SBA`, `ABA`, `PBA1`, and `PBA2`.

6.1. Preliminaries

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 synthetic and real-life datasets from diverse application domains. The main characteristics of the datasets used are briefly described as follows:

—The FOREST COVER (FC) dataset[1] contains 581,012 forest land cells containing 55 attributes. The first 10 numeric attributes have been used, representing positions, distances to roads, hydrology, and so forth. The distance function used is the Euclidean (L_2) distance.
—The ZILLOW (ZIL) dataset[2] contains real estate data, used also in Vlachou et al. [2008]. It contains more than 2M records, but we selected the 1,224,406 records with nonzero or empty values in all their attributes. The dataset has five attributes with the following order: number of bathrooms, number of bedrooms, living area, price, and lot area. The Euclidean distance was also used as the distance function for this dataset.
—The CALIFORNIA (CAL) road network[3] contains 1,965,206 nodes and 5,533,214 edges. The average node degree is equal to 5.63, the average edge weight is 0.016,

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

and the diameter (maximum shortest path distance) is 16.42. The distance function used is the shortest path between network nodes.
—The SAN FRANCISCO (SF) road network[4] contains 174,956 nodes and 223,001 edges. The average node degree is equal to 2.55, the average edge weight is 8.78, and the diameter (maximum shortest path distance) is 16,828.54. The distance function used is the shortest path between network nodes.
—The PROTEIN (PR) dataset contains 346,481 atom sequences. The atom sequences were generated using a sliding window of size 20 applied on 120 large protein structures, downloaded from the RCSB PDB Protein Data Bank.[5] The distance function used is the Levenshtein metric (edit distance).
—For comparison purposes, we have also used a synthetic dataset (UNI), which contains 1,000,000 four-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 dataset 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 dataset. The larger the c value, the more distant the query objects contained in Q. Unless otherwise specified, the parameters take the following values: (1) number of query objects (m) ranges between two and 20 with a default value of five; (2) query coverage (c) takes the values 1%, 5%, 10%, 20%, 30%, 50%, and 100% with a default value of 20%; and (3) the number of results (k) ranges between one and 30 with a default value of 10.

There are three basic criteria used to evaluate and compare the performance of the proposed algorithms: (1) the CPU time required for computations, (2) the I/O time devoted for disk accesses, and (3) the number of distance computations required. It is important to note that for the 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 retrieval of the first results without the need for waiting for the computation of the complete set of answers.

6.2. Experimental Comparison of Algorithms

In this section, we report results related to the overall time required by the algorithms to provide all k results. In addition, we give in a separate subsection results regarding the distance computations required by the algorithms.

6.2.1. Impact of Aggregation Functions on ABA Performance. An issue that requires our attention is the type of the aggregation function utilized by `ABA`. Recall that according to Lemma 4.2, `ABA` works with any monotone aggregation function. For this reason, we experimented with four different aggregation functions: `sum`, `avg`, `min`, and `max`, all of which satisfy the monotonicity property. Table II depicts the results of this comparison among the four aggregates for each dataset, and we have highlighted the best-performing one for each case. The `sum` aggregation has the best performance in all cases, except for the ZIL dataset, where `max` performs the best.

The second remark is that `min` and `max` have a significant difference in their runtime performance. We also observe a clear trend between the two functions: in the case of road networks (datasets CAL and SF), `min` is better than `max`, whereas in the remaining cases, the opposite is true. In principle, with the `min` aggregation, the first radius is

[4]http://www.rtreeportal.org.
[5]http://www.rcsb.org/pdb/home/home.do.

Table II. Performance Analysis of ABA with Different Aggregation Functions, for the Various Datasets

Algorithm	UNI	FC	ZIL	CAL	SF	PR
ABA-sum	439.134	414.30	418.22	17635.74	3995.75	140.17
ABA-avg	569.27	431.02	463.59	17975.67	5170.63	142.94
ABA-min	1023.17	547.77	637.66	18545.51	4342.01	215.53
ABA-max	504.23	295.96	434.14	37406.48	6767.49	166.42

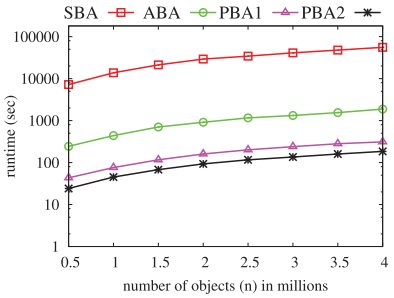

Fig. 6. Running time versus number of objects (n).

small, whereas the subsequent ones are large. This has a heavy impact on the number of disk accesses, due to the large radii in range queries, performed by ABA-min. On the contrary, ABA-max does not face this problem, because the max aggregation function makes the first radius to be large, thereby saving disk accesses compared to min. However, this is not the case for datasets CAL and SF, where the distance between objects is given by their shortest path. The shortest path is a costly operation computationally wise and, as we will demonstrate in the next paragraphs, it can dominate the overall runtime. When using the max aggregation, the first radius is large, which may reduce I/O cost, but includes more objects in the range queries, which leads to more distance computations. This, consequently, degrades performance, given that the CPU cost is higher than the I/O cost for these datasets, and max performs worse than min, which constraints distance computations.

Based on this preliminary performance analysis and the fact that the sum aggregation takes into account distances in a natural way—much more than the alternatives—we will use it (the sum) as the default variation of ABA in the subsequent experimental evaluation.

6.2.2. Runtime Results. In this section, we report results related to the overall time required by the algorithms to provide all k results. We begin by evaluating the performance of the four algorithms when we vary the number of objects in the dataset n. For this particular experiment, we generated synthetic datasets of varied cardinality, with uniform distribution in the 4D space, following the same process through which the UNI dataset was created. Figure 6 depicts the total running time of all four algorithms for this experiment. We observe a linear relation between the total runtime and the number of objects in all cases. PBA1 and PBA2 have clearly and consistently the best overall performance, which renders them more suitable for large datasets.

We now proceed to compare the proposed algorithms by varying the cardinality of the query objects m while keeping the other parameters to their default values. Figure 7 depicts these performance results. We observe that as we increase m, the performance cost increases as well for all algorithms. SBA has the worst performance of all techniques, and its cost increases substantially, even with a few query objects. More specifically,

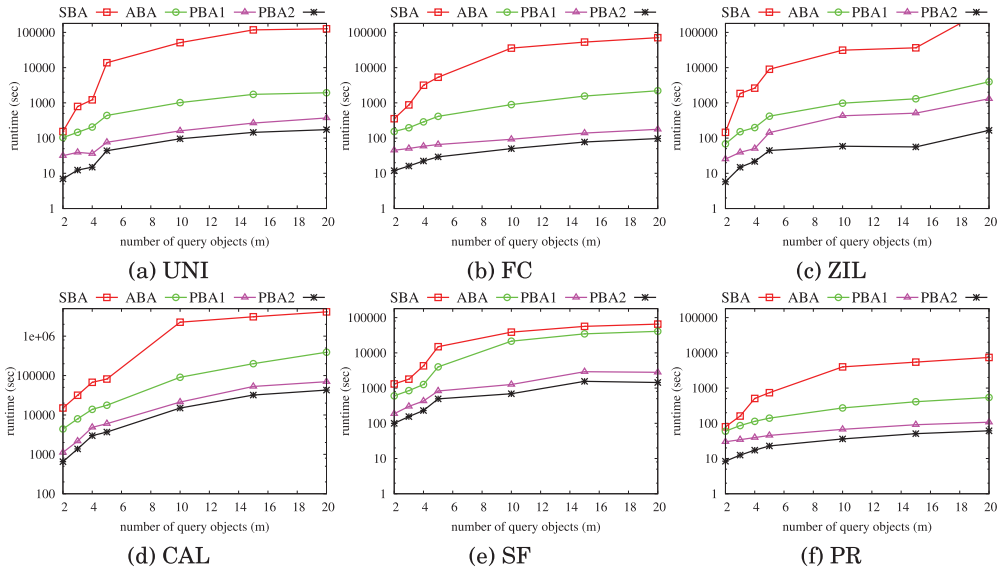

Fig. 7. Running time versus number of queries (m).

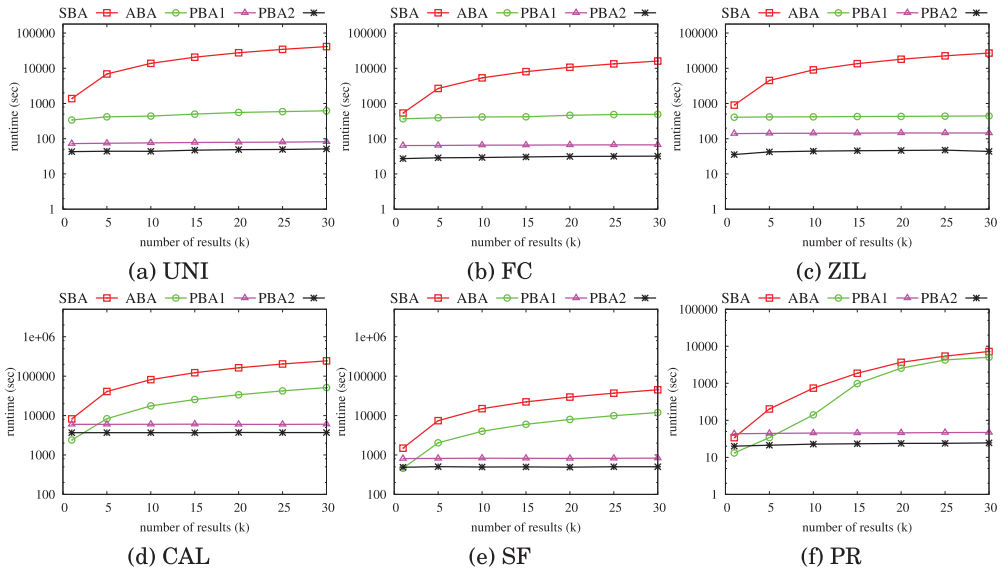

Fig. 8. Running time versus number of results (k), $1 \leq k \leq 30$.

for $m > 5$, we observe that the runtime of SBA increases by an order of magnitude. The other three algorithms have a more stable behavior for the various values of m, across all datasets, with PBA2 performing clearly better than the rest.

In the following set of experiments, we compare the algorithms by varying the number of retrieved results (k). Figures 8 and 9 depict the corresponding comparative evaluation. In particular, Figure 8 depicts the performance of all algorithms when the number of retrieved results is low, whereas Figure 9 demonstrates the performance for larger values of k (up to 1,000). We also note that in Figure 9, the area coverage

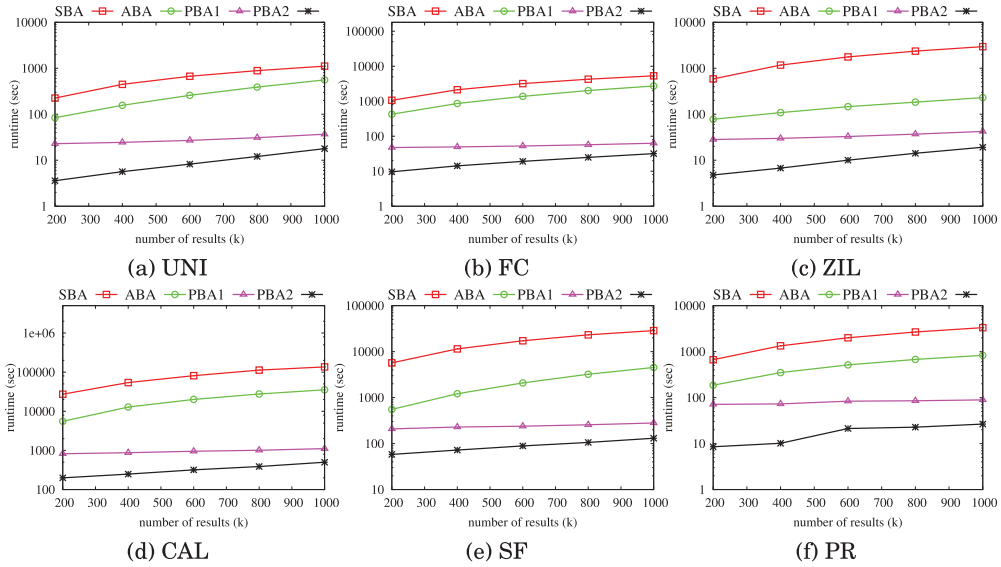

Fig. 9. Running time versus number of results (k), $200 \leq k \leq 1{,}000$.

parameter (c) has been set to 2%, contrary to the 20% default value, in order to demonstrate the behavior of the algorithms for smaller coverage values.

We observe that as we increase k, the computational cost of SBA and ABA increases significantly. The reason is that these two algorithms perform a lot of recalculations using the same objects, because they each spawn a new top-1 dominating query (as shown in Algorithm 1 and Algorithm 2) until they retrieve k results. We also observe that the performance of SBA and ABA is more sensitive to k, whereas PBA1 and PBA2 do not suffer from this problem. This behavior is observed for both small (Figure 8) and large (Figure 9) values of k. PBA1 and PBA2 are not faced with this problem because as soon as they detect the first object, the pruning rules take effect and the remaining top-k results are more easily discovered. This reduces unnecessary computations and I/O cost. On the contrary, SBA and ABA cannot use information from previous iterations to reduce computational costs, as they always search for the top-1 dominating object. It is also worth noting that once again, PBA2 outperforms the other algorithms consistently and significantly.

Figure 10 depicts the comparative performance of the four algorithms when we vary the query coverage c parameter. Query coverage determines how close the randomly selected query objects will be to each other. By increasing this parameter, the number of query objects is kept constant, but the distances among them become larger and we effectively obtain a spatial anticorrelation, thereby increasing the cardinality of the metric space skyline. As a result, SBA incurs significant I/O costs, which explains why it shows the worst performance. In contrast, ABA performs better due to fewer I/O costs but is still worse than PBA1 and PBA2, both of which outperform the alternatives significantly, by one to three orders of magnitude.

It is worth noting that SBA performs better than ABA for very low c values and datasets UNI, FC and ZIL. Low c values imply a spatial correlation of the query, making distances from the query objects more or less the same. As a result, the minimum enclosing ball of the query set (used by algorithm B^2MS^2) produces upper bounds that can prune a significant amount of objects and reduce I/O costs significantly. However, this is not the case for CAL and SF, where distance computations are far more costly

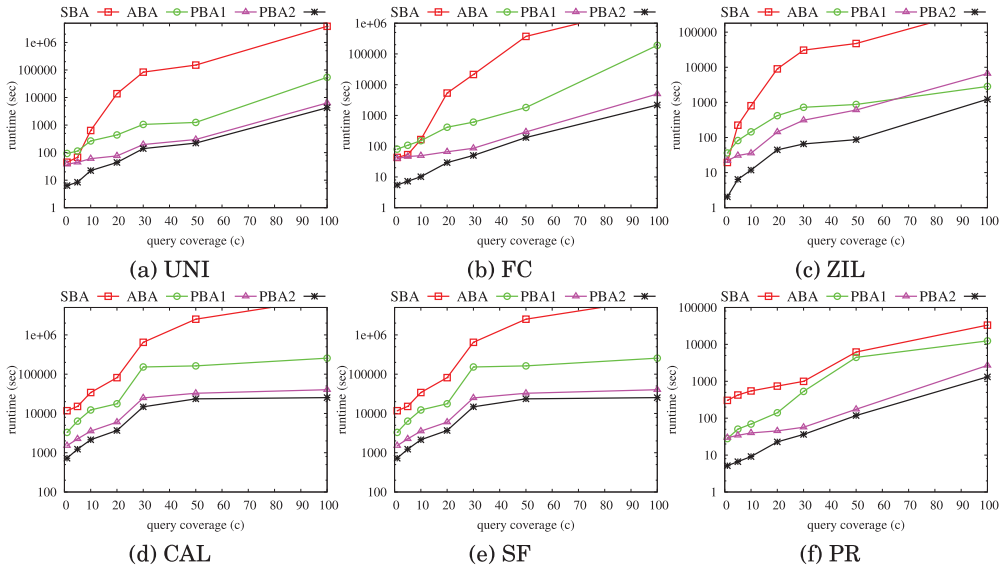

Fig. 10. Running time versus query coverage (c).

Table III. CPU and I/O Cost (In Seconds) for PBA2

Data	Number of Query Objs (m) ($k = 10, c = 20$)			Number of Results (k) ($m = 5, c = 20$)			Query Coverage (c) ($m = 5, k = 10$)		
	2	**5**	**10**	**5**	**10**	**20**	**1%**	**10%**	**20%**
UNI CPU	0.18	11.61	52.52	11.12	11.61	13.84	0.44	3.25	11.61
I/O	6.77	32.22	44.84	31.97	32.22	35.21	5.93	18.92	32.22
FC CPU	0.24	2.83	12.54	2.65	2.83	3.32	0.21	0.43	2.83
I/O	11.62	26.43	37.54	26.09	26.43	28.07	5.24	9.76	26.43
ZIL CPU	0.05	7.54	16.94	5.50	7.54	9.41	0.03	0.46	7.54
I/O	5.71	36.89	41.83	36.87	36.89	36.91	2.01	11.33	36.89
CAL CPU	625.80	3637.64	14828.23	3627.67	3637.64	3669.07	714.01	2111.09	3637.64
I/O	26.00	47.62	140.66	59.36	47.62	59.37	11.34	32.07	47.62
SF CPU	96.11	483.89	681.92	480.49	483.89	495.17	9.91	312.77	483.89
I/O	3.67	10.55	9.44	9.69	10.55	13.02	0.25	5.74	10.55
PR CPU	1.41	6.68	12.47	5.84	6.68	7.05	2.12	3.25	6.68
I/O	6.27	16.06	23.55	15.45	16.06	16.71	3.02	5.87	16.06

than I/Os and dominate the total runtime. Distance computations are unavoidable to establish whether an item is outside the minimum enclosing ball, making SBA the worst-performing alternative for those cases.

6.2.3. Results on Distance Computations. In various applications, such as protein-to-protein interactions, road networks, and social graphs, a single distance computation may be more computationally intensive than several I/O operations. In situations like that, the number of distance computations affects query response time significantly, and is the dominant cost. To better illustrate this fact, consider Table III, which reports the CPU and I/O costs of PBA2 for the various datasets. The highlighted entries for CAL and SF clearly demonstrate that I/O cost is far less significant compared to CPU time, with the latter dominating the overall runtime. Therefore, it is important to minimize distance computations as much as possible toward efficient query processing.

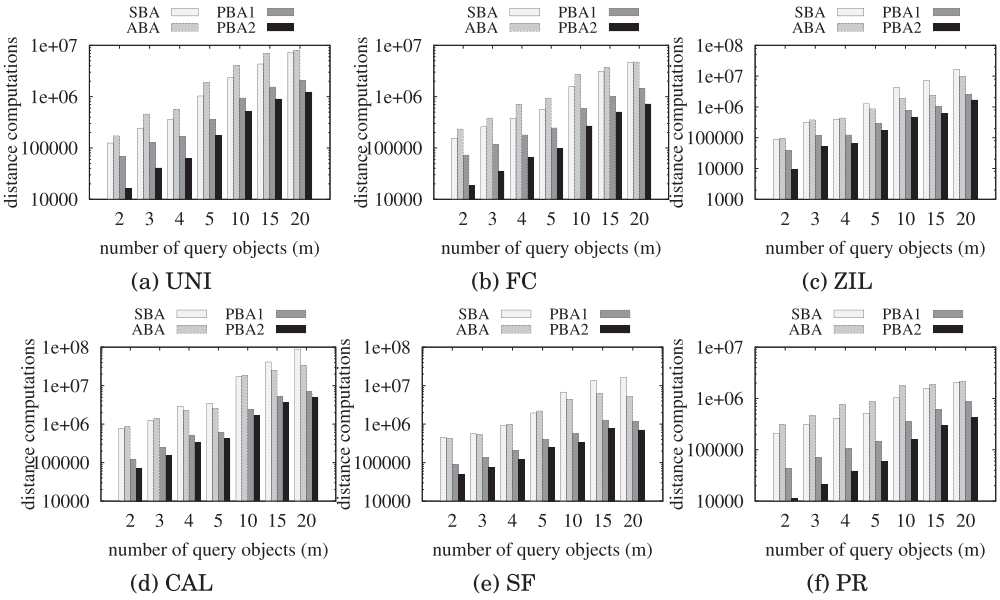

Fig. 11. Number of distance computations versus number of query objects (m).

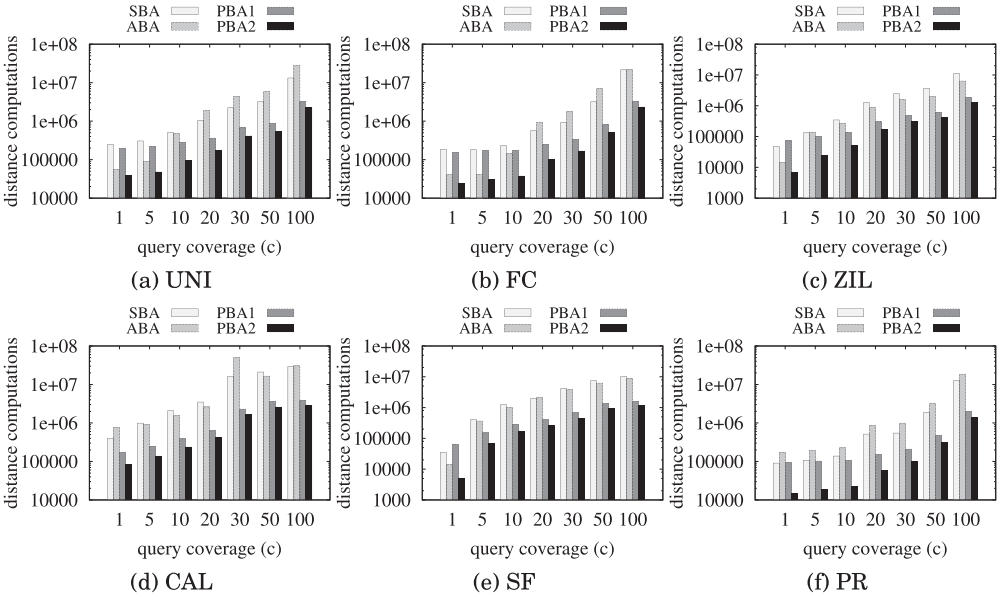

Fig. 12. Number of distance computations versus query coverage (c).

For this reason, we also measure the number of distance computations invoked by the algorithms, which we portray in Figures 11 and 12 for all datasets. More specifically, Figure 11 shows the number of computed distances as a function of the number of query objects m, whereas Figure 12 depicts distance computations compared to query coverage c. Among the proposed algorithms, PBA2 consistently performs the fewest distance computations. This behavior is attributed to both the employed pruning rules—also

Table IV. Number of Exact Score Computations for PBA1 and PBA2

Data	Number of Query Objs (m) ($k = 10, c = 20$)					Number of Results (k) ($m = 5, c = 20$)				Query Coverage (c) ($m = 5, k = 10$)			
	2	**5**	**10**	**15**	**20**	**5**	**10**	**20**	**30**	**1%**	**10%**	**20%**	**50%**
UNI	25	26	26	31	34	16	26	49	77	20	25	26	29
FC	24	25	26	26	26	12	25	49	69	22	22	25	30
ZIL	26	125	158	192	60	85	125	184	231	22	31	125	51
CAL	263	282	55	61	61	229	282	332	363	273	97	282	285
SF	278	87	366	67	575	73	87	107	125	22	243	87	1269
PR	18	18	20	20	18	9	18	37	53	17	17	18	26

shared by PBA1—and the way the $AuxB^{+}$-tree structure is exploited to compute the exact score of objects.

Note that for datasets CAL and SF, PBA1 and PBA2 behave very much in the same way, whereas there is a significant difference in the number of distance computations in the other datasets. By cross-referencing these results with the corresponding ones in Figures 7 and 10, we observe that PBA1 and PBA2 have very similar performances. In other words, the performance boost for these techniques comes from greatly reducing CPU costs, which is the dominant factor, and the steady difference in runtime between them can be attributed to the steady difference of distance computations. On the contrary, distance computations alone cannot account for the performance improvement of PBA1 and PBA2 over SBA and ABA for the other datasets. For example, PBA1 performs more computations than ABA for low c values but has a lower total runtime. Therefore, PBA1 and PBA2 are able to reduce both I/O and CPU costs. Unlike SBA, which succeeds in doing so in a very limited number of cases, the PBA variations can greatly minimize the dominant cost factor in the general case, whether it is I/O or CPU. This makes them well suited for a very broad spectrum of applications.

6.3. More Results on the Pruning-Based Algorithms

Our previous set of experiments showed clearly that PBA1 and PBA2 outperform SBA and ABA in all cases, by at least an order of magnitude. For this reason, in the next series of experiments, we will focus on these two algorithms alone, to better study their performance and behavior.

An important factor unique to PBA1 and PBA2 that affects query execution time is the number of exact score computations performed by these techniques. Recall that, unlike SBA and ABA, the two PBA variations can establish an estimation of the dominance score of an object and may reject it immediately without computing its exact score. Therefore, we would like to know the number of exact computations that PBA1 and PBA2 perform, which will allow us to better analyze their internal behavior.

Table IV reports on the performance of the two algorithms for this particular measure, for all datasets. We can immediately observe that the number of exact computations is only a small fraction of the datasets' cardinality, which is one of the main ingredients for the excellent performance of the two techniques. With a more careful inspection, we also observe that the number of dominance score computations grows, as a general trend, when we increase the number of query objects, the number of results, and the query coverage. There are some cases that contradict this trend, which can be attributed to dataset characteristics, selected query objects, and selected pruning rules. As an example, we may have to perform more exact score computations when dealing with two query points ($m = 2$), compared to having five query points ($m = 5$), which is the case for the SF dataset. Although this may seem counterintuitive, in the latter case we are able to eliminate more objects through our pruning rules compared to the former one, which then needs to compute the exact scores of more points, to identify

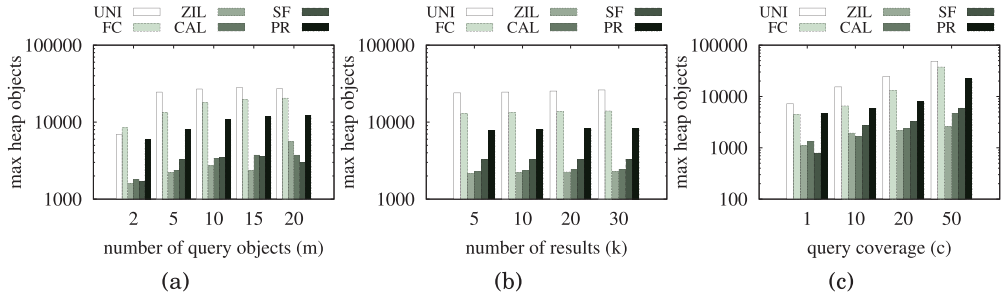

Fig. 13. Maximum number of objects accommodated in main memory for algorithms PBA1 and PBA2.

the top-k result. Despite these fluctuations, the total number of exact computations is always kept to a minimum.

Main memory consumption is also an important performance index in query processing, especially when we consider the memory used to accommodate temporary data. In our case, a heap data structure is used to store some objects that are potential candidates for the next best object. In addition, the incremental nearest-neighbor mechanism over the M-tree access methods maintains temporary data in main memory. Figure 13 depicts the maximum number of objects maintained in main memory during query processing. It is evident that the number of objects is small compared to the dataset cardinality, and therefore PBA2 does not pose a significant storage overhead. However, even in cases where the heap capacity is limited, efficient secondary storage implementations [Fadel et al. 1999] may be employed.

Pruning rules are another distinctive characteristic of PBA1 and PBA2 and play an integral part in their overall performance, as we have already pointed out. In the sequel, we will focus on the effectiveness of the employed pruning strategies, and more specifically on the number of objects that they eliminate. In order to evaluate the pruning power of each particular family, we consider it independently of the others. In other words, we perform a set of experiments where only one family of rules is activated at a time, whereas the other two are disabled. We also note that in this analysis, we will only consider objects that have been inserted in the $AuxB^+$-tree and are potential candidates for the top-k result. The reason for focusing on these items alone is that, unlike items that are immediately rejected, objects inserted in the $AuxB^+$-tree consume resources (e.g., memory). Therefore, we are interested in the pruning power of each family of rules on the items that introduce an additional overhead. However, we stress that the vast majority of objects are actually discarded without consideration. To give an idea of this fact, we mention that for the CAL dataset, in the case where $m = 10$, $k = 10$, and $c = 20$, from a total of 1,965,206 objects, 1,824,443 (more than 90%) of them are completely ignored by PBA1 and PBA2.

The effectiveness of the pruning rules is shown in Figure 14, where we focus on CAL. Similar results—including the high percentage of pruned points without consideration—were obtained for the other datasets. The first observation is that a significant number of objects is eliminated by the pruning mechanisms: of the remaining 140K objects, a total of approximately 10% is pruned by the rules. However, the number of the eliminated objects per rule family may differ depending on several factors. For example, in Figure 14(a), it is observed that the number of query objects has a significant impact on pruning effectiveness. For a few query objects m, DPR and EPR eliminate more objects than IPR, whereas by increasing the number of query objects, IPR's effectiveness improves significantly. This can be explained if we consider that by increasing the number of query objects, which effectively increases the problem

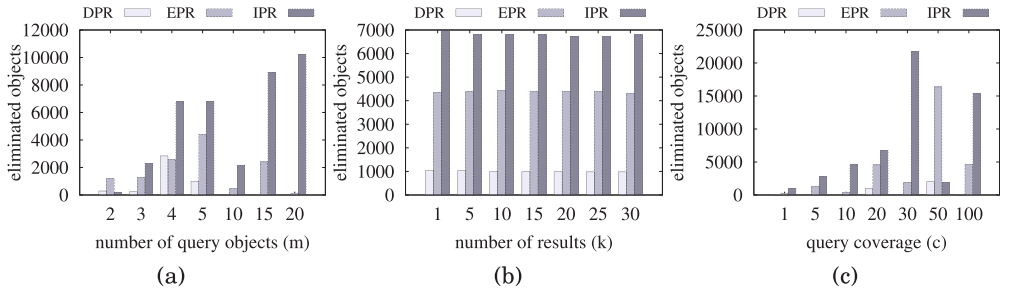

Fig. 14. Effectiveness of pruning rules in algorithm PBA2: number of eliminated objects for CAL dataset versus number of query objects (m), number of results k, and query coverage (c).

dimensionality, it becomes more difficult to prune away objects through dominance checks, because more points belong to the skyline. Consequently, pruning becomes restricted to rules that operate on the dominance score, such as IPR and some rules of the EPR family. The sophistication of EPR allows it to continue pruning objects even for larger values of m. A similar behavior is observed when we vary the query coverage in Figure 14(c). On the other hand, pruning effectiveness is almost steady by increasing the number of results, as is depicted in Figure 14(b).

In conclusion, the pruning-based algorithms PBA1 and PBA2 perform orders of magnitude better than 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. The good performance of PBA variations is attributed to the following reasons:

—The incremental nearest-neighbor technique applied in conjunction with the round-robin examination of distances from query objects saves a significant number of distance computations, due to the fact that it works in a way similar to Fagin's TA, which has the instance optimality property.
—The use of the auxiliary B-tree access method manages to reduce the number of I/Os several orders of magnitude in comparison to the baseline algorithms that use a significant number of iterations to provide the result. This effect has a direct impact on performance.
—The pruning rules help to decrease the computational costs further since they eliminate objects saving exact score computations.

6.4. Toward an Approximate Solution

Based on the previous results, we observe that the performance of all metric-based top-k dominating algorithms is strongly dependent on the number of query objects (m). Every query object forces the execution of an incremental nearest-neighbor search over the M-tree structure. Therefore, the greater the number of query objects, the more costly processing becomes (e.g., see Figure 7).

Motivated by this behavior, we performed some additional experiments toward an approximate solution, whereby we sacrifice the accuracy of the results but reduce computational costs significantly. Assume, for instance, that $m = 2$; that is, there are only two query objects. It is natural to deduce that if two objects are sufficiently close to each other, which is what we obtain with a low query coverage c value, then it may be possible to group them together. This would be an interesting alternative to the aforementioned algorithms provided that the loss of accuracy is not significant. A simple way to quantify the accuracy is to measure the percentage of the objects that

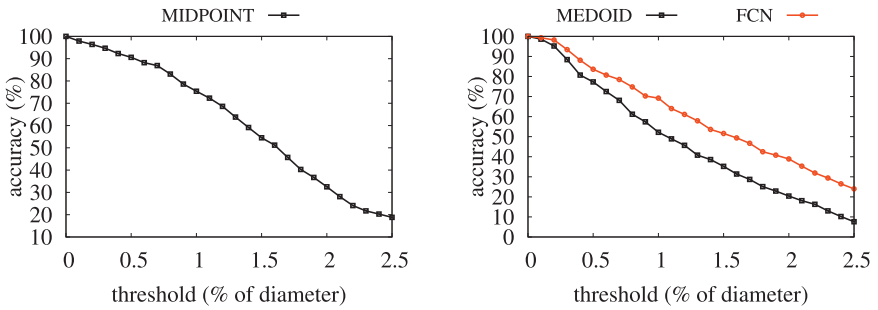

Fig. 15. Accuracy versus the percentage of the diameter as a grouping threshold for $m = 2$ query objects.

are retrieved by the approximate algorithm compared against the result set consisting of the exact solution.

In order to evaluate the accuracy/speed tradeoff, we have performed a series of experiments using the UNI and the CAL datasets. In the case of UNI, we replace the two query objects with their *midpoint* and we gradually increase their pairwise distance. The achieved accuracy as a function of the distance between the query objects is illustrated in Figure 15(a). We observe that when the distance between the query points is less than 1% of the dataset diameter, the accuracy is more than 70%.

Similar results are obtained for the CAL dataset. Since CAL is a network, the concept of midpoint is not defined. Instead, we have used two alternatives to group query objects: (1) we have used the *medoid* along the shortest path joining the query objects, and (2) we have used the *first common neighbor* (FCN) as their representative. The results for both cases are shown in Figure 15(b). Again, for distances less than 1% of the diameter, the accuracy is more than 60%. In this experiment, we also observe that the grouping performed by the FCN alternative performs better than that of selecting the medoid.

The previous discussion leads naturally to an approximation algorithm that generalizes the grouping idea of query objects. Instead of grouping pairs of objects, an initial clustering may form larger groups that lie within the grouping threshold (e.g., 1% of the diameter). It is interesting to study in detail this approach and provide approximation guarantees regarding the error introduced in the result, and also to apply more sophisticated ways (e.g., rank correlation) to measure accuracy taking into account the rank position of false positives and false negatives.

7. CONCLUSIONS

Top-k dominating queries combine the benefits of skyline and general top-k query processing. The ranking supported by this query type is based on the number of dominated objects. Therefore, an object x receives a high score if x dominates a large number of objects. This article 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, and the third and fourth one (`PBA1` and `PBA2`) use incremental nearest-neighbor search equipped by (1) a set of effective pruning rules to reduce the search space and (2) 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. In Section 6.4, we gave some preliminary results toward this direction, showing that it is possible to group query objects if they are sufficiently close to each other. Providing error guarantees is very important and it is expected to be a difficult task, taking into account that the analysis of top-k dominating queries is not straightforward. In fact, recently Kosmatopoulos et al. [2014] have shown that even for the 2-dimensional case, the worst-case complexity analysis is not trivial. Therefore, it is very interesting to provide analytical results for both the exact and the approximate solution.

Another interesting extension is to consider the problem in a parallel/distributed setting, offering significant scalability, especially for massive datasets.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their helpful comments and suggestions, as well as the PC chairs of EDBT 2014 for selecting our paper as a candidate for the ACM TODS Special Issue.

REFERENCES

Wolf-Tilo Balke, Ulrich Gntzer, and Jason Xin Zheng. 2004. Efficient distributed skylining for web information systems. In *EDBT*. 256–273.

John Louis Bentley, Hsiang-Tsung Kung, Mario Schkolnick, and C. D. Thompson. 1978. On the average number of maxima in a set of vectors and applications. *J. ACM* 25, 4 (1978), 536–543.

Stephan Börzsönyi, Donald Kossmann, and Konrad Stocker. 2001. The skyline operator. In *Proceedings of ICDE'01*. 421–430.

Tolga Bozkaya and Meral Ozsoyoglu. 1999. Indexing large metric spaces for similarity search queries. *ACM Trans. Database Syst.* 24, 3 (Sept. 1999), 361–404.

Sergey Brin. 1995. Near neighbor search in large metric spaces. In *Proceedings of the 21th International Conference on Very Large Data Bases (VLDB'95)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, 574–584.

Edgar Chávez, Gonzalo Navarro, Ricardo Baeza-Yates, and José Luis Marroquín. 2001. Searching in metric spaces. *ACM Comput. Surv.* 33, 3 (Sept. 2001), 273–321.

Lei Chen and Xiang Lian. 2008. Dynamic skyline queries in metric spaces. In *Proceedings of the 11th International Conference on Extending Database Technology: Advances in Database Technology (EDBT'08)*. ACM, New York, NY, 333–343.

Lei Chen and Xiang Lian. 2009. Efficient processing of metric skyline queries. *IEEE Trans. Knowl. Data Eng.* 21, 3 (2009), 351–365.

Paolo Ciaccia, Marco Patella, and Pavel Zezula. 1997. M-tree: An efficient access method for similarity search in metric spaces. In *Proceedings of the 23rd International Conference on Very Large Data Bases (VLDB'97)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, 426–435.

J. Shane Culpepper, Matthias Petri, and Falk Scholer. 2012. Efficient in-memory top-k document retrieval. In *Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'12)*. 225–234.

Atish Das Sarma, Ashwin Lall, Danupon Nanongkai, and Jun Xu. 2009. Randomized multi-pass streaming skyline algorithms. *Proc. of VLDB Endowment* 2, 1 (Aug. 2009), 85–96.

Ke Deng, Xiaofang Zhou, and Tao Shen. 2007. Multi-source skyline query processing in road networks. In *Proceedings of the 23rd International Conference on Data Engineering (ICDE'07)*. 796–805.

Cynthia Dwork, Ravi Kumar, Moni Naor, and D. Sivakumar. 2001. Rank aggregation methods for the web. In *Proceedings of the 10th International Conference on World Wide Web (WWW'01)*. 613–622.

R. Fadel, K. V. Jakobsen, Jyrki Katajainen, and Jukka Teuhola. 1999. Heaps and heapsort on secondary storage. *Theor. Comput. Sci.* 220, 2 (June 1999), 345–362.

Ronald Fagin, Amnon Lotem, and Moni Naor. 2001. Optimal aggregation algorithms for middleware. In *Proceedings of the 20th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS'01)*. ACM, New York, NY, 102–113.

David Fuhry, Ruoming Jin, and Donghui Zhang. 2009. Efficient skyline computation in metric space. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology (EDBT'09)*. ACM, New York, NY, 1042–1051.

Gisli R. Hjaltason and Hanan Samet. 1995. Ranking in spatial databases. In *Proceedings of the 4th International Symposium on Advances in Spatial Databases (SSD'95)*. Springer-Verlag, London, UK, 83–95.

Gisli R. Hjaltason and Hanan Samet. 2003. Index-driven similarity search in metric spaces (survey article). *ACM Trans. Database Syst.* 28, 4 (2003), 517–580.

Vagelis Hristidis, Nick Koudas, and Yannis Papakonstantinou. 2001. PREFER: A system for the efficient execution of multi-parametric ranked queries. In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data (SIGMOD'01)*. ACM, New York, NY, 259–270.

Ihab F. Ilyas, Walid G. Aref, and Ahmed K. Elmagarmid. 2004. Supporting top-k join queries in relational databases. *VLDB J.* 13, 3 (2004), 207–221.

Ihab F. Ilyas, George Beskales, and Mohamed A. Soliman. 2008. A survey of top-k query processing techniques in relational database systems. *Comput. Surv.* 40, 4, Article 11 (2008), 11:1–11:58.

Maria Kontaki, Apostolos N. Papadopoulos, and Yannis Manolopoulos. 2012. Continuous top-k dominating queries. *IEEE Trans. Knowl. Data Eng.* 24, 5 (May 2012), 840–853.

Andreas Kosmatopoulos, Apostolos N. Papadopoulos, and Kostas Tsichlas. 2014. Dynamic processing of dominating queries with performance guarantees. In *Proceedings of the 17th International Conference on Database Theory (ICDT), Athens, Greece, March 24–28, 2014*. 225–234.

Iosif Lazaridis and Sharad Mehrotra. 2001. Progressive approximate aggregate queries with a multi-resolution tree structure. In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data (SIGMOD'01)*. ACM, New York, NY, 401–412.

Xiang Lian and Lei Chen. 2009. Top-k dominating queries in uncertain databases. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology (EDBT'09)*. ACM, New York, NY, 660–671.

Amélie Marian, Nicolas Bruno, and Luis Gravano. 2004. Evaluating top-k queries over web-accessible databases. *ACM Trans. Database Syst.* 29, 2 (2004), 319–362.

Dimitris Papadias, Yufei Tao, Greg Fu, and Bernhard Seeger. 2005a. Progressive skyline computation in database systems. *ACM Trans. Database Syst.* 30, 1 (March 2005), 41–82.

Dimitris Papadias, Yufei Tao, Kyriakos Mouratidis, and Chun Kit Hui. 2005b. Aggregate nearest neighbor queries in spatial databases. *ACM Trans. Database Syst.* 30, 2 (June 2005), 529–576.

Katerina Raptopoulou, Apostolos N. Papadopoulos, and Yannis Manolopoulos. 2003. Fast nearest-neighbor query processing in moving-object databases. *Geoinformatica* 7, 2 (2003), 113–137.

Nick Roussopoulos, Stephen Kelley, and Frédéric Vincent. 1995. Nearest neighbor queries. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data (SIGMOD'95)*. 71–79.

Mehdi Sharifzadeh and Cyrus Shahabi. 2006. The spatial skyline queries. In *Proceedings of the 32nd International Conference on Very Large Data Bases (VLDB'06)*. VLDB Endowment, 751–762.

Cheng Sheng and Yufei Tao. 2011. On finding skylines in external memory. In *Proceedings of the 30th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS'11)*. 107–116.

Dimitrios Skoutas, Dimitris Sacharidis, Alkis Simitsis, Verena Kantere, and Timos Sellis. 2009. Top-k dominant web services under multi-criteria matching. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology (EDBT'09)*. ACM, New York, NY, 898–909.

Eleftherios Tiakas, Apostolos N. Papadopoulos, and Yannis Manolopoulos. 2011. Progressive processing of subspace dominating queries. *VLDB J.* 20, 6 (Dec. 2011), 921–948.

Eleftherios Tiakas, George Valkanas, Apostolos N. Papadopoulos, Yannis Manolopoulos, and Dimitrios Gunopulos. 2014. Metric-based top-k dominating queries. In *Proceedings of the 17th International Conference on Extending Database Technology (EDBT), Athens, Greece, March 24–28, 2014*. 415–426.

Akrivi Vlachou, Christos Doulkeridis, and Yannis Kotidis. 2008. Angle-based space partitioning for efficient parallel skyline computation. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data (SIGMOD'08)*. 227–238.

Yingqi Xu, Tao-Yang Fu, Wang-Chien Lee, and Julian Winter. 2007. Processing K nearest neighbor queries in location-aware sensor networks. *Signal Proc.* 87, 12 (2007), 2861–2881.

Man Lung Yiu and Nikos Mamoulis. 2007. Efficient processing of top-k dominating queries on multi-dimensional data. In *Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB'07)*. VLDB Endowment, 483–494.

Man Lung Yiu and Nikos Mamoulis. 2009. Multi-dimensional top-k dominating queries. *VLDB J.* 18, 3 (June 2009), 695–718.

Wenjie Zhang, Xuemin Lin, Ying Zhang, Jian Pei, and Wei Wang. 2010. Threshold-based probabilistic top-k dominating queries. *VLDB J.* 19, 2 (April 2010), 283–305.

Received February 2015; revised August 2015; accepted October 2015