Top-k Dominating Queries: a Survey

Eleftherios Tiakas
Department of Informatics
Aristotle University
54124 Thessaloniki, Greece
Email: tiakas@csd.auth.gr

Apostolos N. Papadopoulos
Department of Informatics
Aristotle University
54124 Thessaloniki, Greece
Email: papadopo@csd.auth.gr

Yannis Manolopoulos
Department of Informatics
Aristotle University
54124 Thessaloniki, Greece
Email: manolopo@csd.auth.gr

***Abstract*—Top-k dominating queries combine the advantages of top-k queries and skyline queries, and eliminate their disadvantages. They return k objects with the highest domination score, which is defined as the number of dominated objects. As a top-k query, the user can bound the number of returned results through the parameter k, and like a skyline query a user-selected scoring function is not required. Top-k dominating queries have been studied by different perspectives, such as in indexed and non-indexed multi-dimensional data using efficient exact computation algorithms, in uncertain data using randomized algorithms with accuracy guarantees, and in data streams. In addition, top-k dominating queries have been studied over distance-based dynamic attribute vectors, defined over a metric space, using efficient progressive algorithms. Top-k dominating queries have become an important tool for decision support, data mining, web search, and multi-criteria retrieval applications.**

I. Introduction

During the last years, preference-based queries have attracted significant attention. In particular, two query types have gained increasing importance and have been studied extensively: i) top-k queries and ii) skyline queries. In a top-k query, a user provides a ranking score function (usually monotone) to order the objects by their scores and, thus, retrieve the top-k best objects. The skyline is composed of the objects that are not dominated, based on a domination relationship involving the values in each dimension. More formally, the object $p = (p.x_1, p.x_2, \ldots, p.x_d) \in D$ dominates another object $q = (q.x_1, q.x_2, \ldots, q.x_d) \in D$, i.e., $p \prec q$, when: $\forall i \in \{1, \ldots, d\} : p.x_i \leq q.x_i \wedge \exists i \in \{1, \ldots, d\} : p.x_i < q.x_i$. This means that p is as good as q in all dimensions, and it is strictly better than q in at least one dimension. Then, the domination score of p, $dom(p)$ is defined as: $dom(p) = |\{q \in S : p \prec q\}|$. A top-$k$ dominating query returns the k objects with the maximum domination scores in D. For example, the work in [11] exploits this concept for ranking web services.

The most important advantage of a top-k query is that the number of results is bounded (which is not true for skyline queries), whereas the most important advantage of a skyline query is that no parameters and user-defined scoring functions are required. Top-k dominating queries combine the advantages of top-k queries and skyline queries, eliminating their disadvantages by assigning to each object an intuitive score based on dominance. This score is reflecting the importance of every object in the dataset in a natural way.

These concepts are better explained by an example. The objects of the dataset D are the two-dimensional points p_i for i=1,. . .,15. Without loss of generality, we assume that there is preference in small values in all dimensions. Then, the rectangular area that has a greater or equal x value and a greater or equal y value in comparison to the coordinates of a specific point p, it is called the *domination region* of p. The domination region contains all points that p dominates. For example, the domination region of p_6 in Figure 1 contains 6 points, thus $dom(p_6) = 6$. By using domination regions, we can calculate all domination scores of the points p_i, i.e., $dom(p_1) = 10$, $dom(p_2) = 12$, $dom(p_3) = 5$, $dom(p_4) = 10$, $dom(p_5) = 0$, $dom(p_6) = 6$, $dom(p_7) = 5$, $dom(p_8) = 1$, $dom(p_9) = 4$, $dom(p_{10}) = 0$, $dom(p_{11}) = 1$, $dom(p_{12}) = 1$, $dom(p_{13}) = 2$, $dom(p_{14}) = 1$, $dom(p_{15}) = 0$. Therefore, a top-3 dominating query must return the points p_2, p_1, p_4. Note that p_1 and p_2 are skyline points, but p_4 is not.

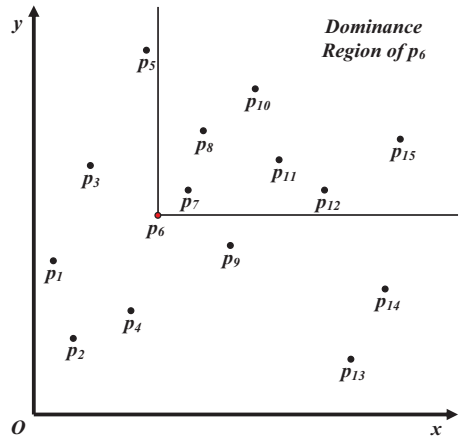

Fig. 1. An example of a dominating query.

II. A Skyline-Based Top-k Dominating General Algorithm (STD)

The first approach for processing top-k dominating queries has been presented in [9], where the top-k dominating queries have been defined as a variation of skyline queries. A Skyline-Based Top-k Dominating Algorithm (STD) has been proposed, which uses the Branch-and-Bound Skyline Algorithm (BBS) of [9]. The main steps taken by STD are the following:

- Compute the skyline S of the dataset D
- For each object $p \in S$ count the number of objects it dominates (i.e., its score $dom(p)$)

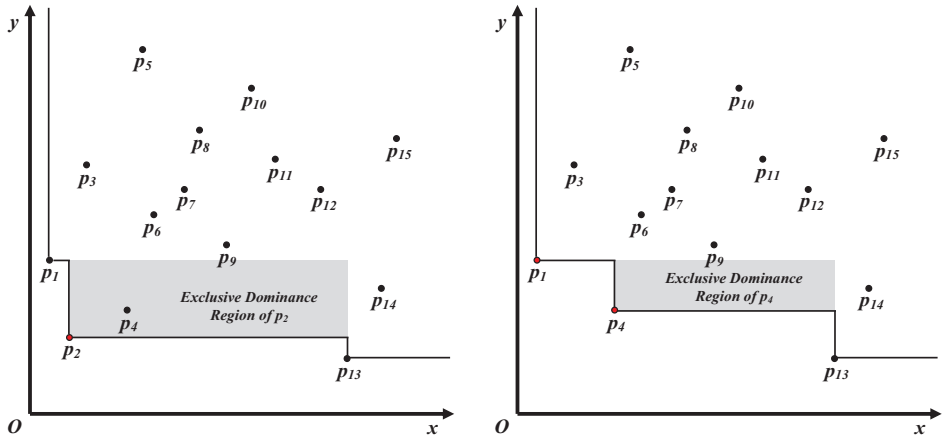

Fig. 2. Example of a top-k dominating query with the STD algorithm.

- Sort the objects in S by their domination scores and report the object q with the maximum domination score (top-1). The domination scores are kept in a sorted list.
- Then, the *exclusive domination region* R of q is selected and a local skyline query constrained in R is performed. The exclusive domination region of an object q of S is the region which contains the objects dominated only by q and not by any other object of the skyline S. After the execution of the constrained skyline query in R, q is removed.
- The domination values of all skyline objects in R are computed and the sorted list is updated.
- The process is repeated for the next top object in the list, until all top-k dominating objects are reported.

We illustrate the previous steps for the dataset depicted in Figure 1. The skyline S of D is S=$\{p_1, p_2, p_{13}\}$. The domination scores of all points in S are computed and inserted in a sorted list: $\{< p_2, 12 >, < p_1, 10 >, < p_{13}, 2 >\}$. Then, the exclusive domination region R of p_2 is selected (see Figure 2a), p_2 is reported as top-1, and then is removed. The constrained skyline in R contains only p_4, where the domination score of p_4 is $dom(p_4)$=10. The sorted list is updated: $\{< p_1, 10 >, < p_4, 10 >, < p_{13}, 2 >\}$. Both points p_1, p_4 are reported as top-2 and top-3, as they have equal domination scores, whereas all other points in the constrained skylines of their exclusive domination regions have less scores. More specifically, the exclusive domination region of p_1 is equal with its full domination region, and the exclusive domination region of p_4 is empty (Figure 2b).

The STD algorithm can be viewed as skyline "peeling", since it computes local skylines among objects that have the highest domination score. Moreover, due to its generality it can be applied in any dataset whereas the computation of the skylines inside STD can be performed by any known off-the-shelf skyline algorithm (i.e., not only the BBS [9]).

III. Top-k Dominating Algorithms on Multi-Dimensional Data

In this section we present algorithms for the efficient processing of top-k dominating queries on indexed multi-dimensional datasets.

A. Methods based on R-Trees and aggregate R-Trees

The STD algorithm can be applied easily in multi-dimensional data. An R-Tree based implementation and extensive experiments can be found in [9]. However, the skyline-based approach may perform many unnecessary score counting, since the skyline could be much larger than k (especially if data have many dimensions).

In [6], [7] another characteristic of a top-k dominating query is revealed. It is in fact an aggregate query, when the data are indexed by using hierarchical index structures (such as R-trees [3]), since computing the domination score $dop(p)$ of a multi-dimensional object $p \in D$ is an aggregation in D. Therefore, the authors replace the R-tree by an aggregate R-tree (aR-tree [5], [8]), and present their methods for the top-k dominating query processing. The aR-tree augments to each non-leaf entry of the R-tree an aggregate measure of all data objects in the subtree pointed by it. It has been used to speed up the evaluation of spatial aggregate queries, where measures in a spatial region are aggregated. As the domination score of an object p is computed by the number of objects that p dominates, a simple count aggregation measure is required for equipping the aR-tree. Therefore, each non-leaf entry stores the count of data objects in its subtree.

Four specialized aR-tree-based algorithms named Simple Counting Guided (SCG), Lightweight Counting Guided (LCG), Upper-bound Based Traversal (UBT), and Cost-Based Traversal (CBT) have been proposed. These algorithms have been compared with STD implemented with a spatial aggregation technique (for fairness in comparison), and with an optimized version of STD named Iterative Top-k Dominating Algorithm (ITD).

The algorithms studied in [6], [7] use specific score bounding functions applied on the nodes of the aR-tree. For an aR-tree entry e (i.e., a minimum bounding box) whose projection on the i-th dimension is the interval $[e[i]^-, e[i]^+]$, its lower corner is denoted by $e^- = (e[1]^-, e[2]^-, \ldots, e[d]^-)$ and its upper corner is denoted by $e^+ = (e[1]^+, e[2]^+, \ldots, e[d]^+)$, where d is the number of dimensions in the dataset D. Both e^- and e^+ do not correspond to actual data objects but they allow us to express domination relationships among objects and minimum bounding boxes conveniently. Figure 3 depicts some examples. The object p_1 dominates the region of entry e_1, thus it dominates all data objects that are indexed under e_1 (full domination). The object p_1 dominates e_1^+ but not e_1^-, thus it dominates some but not all data objects that are indexed under e_1 (partial domination). Finally, the entire domination region of object p_3 is disjoint with the region of entry e_1, thus it cannot dominate any object indexed under e_1 (no domination).

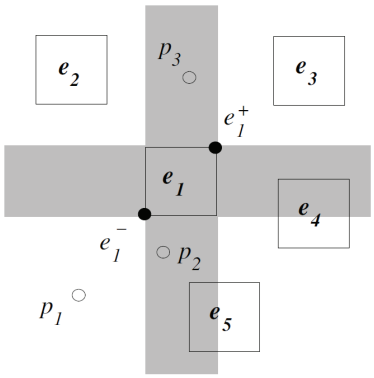

Fig. 3. Example of domination relationship among aR-tree entries.

For any aR-tree entry e, the values $dom(e^+)$ and $dom(e^-)$ correspond to the tight-most lower and upper score bounds respectively, for any object indexed under e.

1) Iterative Top-k Dominating Algorithm (ITD): The Iterative Top-k Dominating Algorithm (ITD [6], [7]) is an optimized version of STD, applied with two optimizations that greatly reduce the I/O cost when the data are indexed with an aR-tree. The first optimization is called "batch counting". Instead of iteratively applying separate range queries to compute the scores of the skyline objects, ITD executes them in a recursive batch counting procedure (Batch-Count) on the aR-Tree. More specifically, when a Batch-Count must be conducted for a node z of the aR-tree, then:

- For all entries e of node z, we check:
- If z is a non-leaf node, and exists an object p of the current candidates that dominates e^+ but not e^-, then the Batch-Count procedure is called recursively for the child node of z.
- Else z is a leaf node, then all domination scores of the objects p in the current candidate set that dominate e^-, are updated by the count of entry e, i.e., $dom(p) = dom(p) + count(e)$.

This procedure calculates only the non-calculated domination values of the skyline objects and only in the exclusive domination regions, recursively, thus avoids recalculations. The second optimization is that the set structure which keeps all current candidate objects is sorted by a space-filling curve (Hilbert ordering) before applying the batch counting to increase the compactness of the MBR of a batch. These two optimizations are greatly reducing the I/O cost of STD. The example of Figure 2 illustrates also the processing of ITD.

However, the skyline based solutions becomes inefficient for datasets with large skylines. Moreover, as the dimensionality of the dataset D increases, the skyline S may become as big as D. An important study in [13] reports that there is a specific dimension (not large), called the "eliminating dimension", where all domination values in D become zero, thus the skyline of D contains all objects of D (S=D).

Motivated by these observations, the following proposed algorithms in [6], [7] solve the problem directly, without depending on skyline computations.

2) Simple Counting Guided Algorithm (SCG): In the aR-tree the score of any object p indexed under an entry e is upper-bounded by $dom(e^-)$. By using this property, the main idea of SCG ([6], [7]) is to traverse the aR-tree nodes in a descending order of their upper-bound scores. The rationale is that objects with high scores can be retrieved early and, thus, accesses to aR-tree nodes that do not contribute to the result can be avoided. To organize the entries to be visited in descending order of their scores, a max-heap structure H is used. The top-k dominating objects are managed by a min-heap structure W as the algorithm progresses, while g is the current k-th score in W used for pruning (any object p with a domination score $dom(p) < g$ must not be appeared in the query results).

More specifically, the algorithm takes the following steps:

- The upper bound scores $dom(e^-)$ of the aR-tree root entries are computed in batch (using the Batch-Count procedure of ITD), and are inserted into the max-heap H.
- While the score $dom(e^-)$ of H's top entry e is higher than g, the top entry is deheaped, and the node z pointed by e is visited.
- If z is a non-leaf node, its entries are enheaped, after Batch-Count is called to compute their upper score bounds.
- If z is a leaf node, the scores of the objects in it are computed in batch and the top-k set W is updated, if applicable.

In the sequel, we illustrate the execution of SCG. For simplicity, we perform a top-1 dominating query. Figure 4 illustrates the processing. In the aR-tree there are 5 leaf nodes and their corresponding entries in the root node are e_1, e_2, e_3, e_4, e_5. First, the upper bound scores for the root entries are computed with the batch-counting procedure, i.e., $dom(e_1^-) = 14$, $dom(e_2^-) = 9$, $dom(e_3^-) = 7$, $dom(e_4^-) = 3$, $dom(e_5^-) = 3$. Since e_1 has the highest upper bound score, the leaf node pointed by e_1 will be accessed next. The scores of entries in e_1 are computed in batch, i.e., $dom(p_1) = 10$, $dom(p_2) = 12$, $dom(p_4) = 10$. Since p_2 has the higher domination score of all remaining entries ($p_1, p_4, e_2, e_3, e_4, e_5$), it is guaranteed to be the top-1 result.

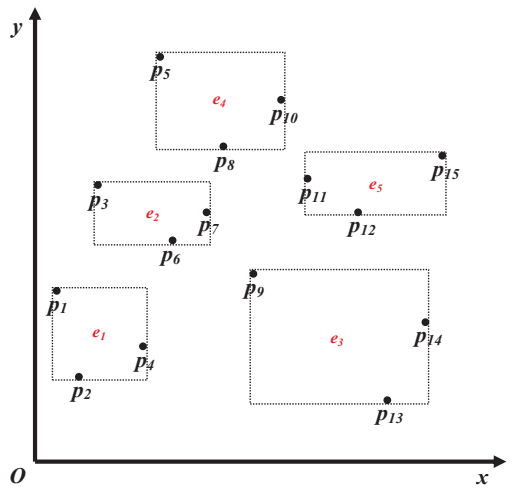

Fig. 4. Example of SCG Algorithm.

3) Lightweight Counting Guided Algorithm (LCG): The Lightweight Counting Guided Algorithm (LCG [6], [7]) is an optimized version of SCG. The main idea of the algorithm is to replace the tight upper score bound $dom(e^-)$ by a looser and cheaper to compute bound $dom^u(e)$. The algorithm processes exactly as SCG with the difference that the Batch-Count procedure is replaced by a Light-Batch-Count procedure, which is a variation of Batch-Count. In specific, when bounds for a set of non-leaf entries are counted, the algorithm avoids expensive accesses at aR-tree leaf nodes, by using entries at non-leaf nodes to derive looser bounds. The example of Figure 4 illustrates also the processing of LCG, with the difference that the following looser bounds are used: $dom^u(e_1) = 15$, $dom^u(e_2) = 9$, $dom^u(e_3) = 9$, $dom^u(e_4) = 3$, $dom^u(e_5) = 3$. The bound are calculated without processing the p_i objects but by using only the aggregation count of nodes e_i. For example, the $dom^u(e_1)$ value is equal to 15 because e_1 fully or partially dominates all other e_i regions, i.e., $dom^u(e_1) = count(e_1)+count(e_2)+count(e_3)+count(e_4)+count(e_5) = 15$. The $dom^u(e_3)$ value is equal to 9 because e_3 fully or partially dominates only the regions e_4, e_5 regions, i.e., $dom^u(e_3) = count(e_3) + count(e_4) + count(e_5) = 9$.

4) Upper-bound Based Traversal Algorithm (UBT): The Counting-Guided approaches, may access some aR-tree nodes more than once due to the application of counting operations for the visited entries. In [6], [7] Priority-Based Traversal Algorithms are proposed, where the general idea is that instead of computing upper bounds of visited entries by explicit counting to defer score computations for entries, and maintain lower and upper bounds for them during the tree is traversed. Score bounds for visited entries are gradually refined when more nodes are accessed, until the result is finalized with the help of them. For this method to be effective, the tree is traversed with a carefully designed priority order aiming at minimizing I/O cost.

The general Priority-Based Traversal Algorithm (PBT), during traversal, maintains a set S of visited aR-tree entries, and loose upper $dom^u(e)$ and lower $dom^l(e)$ score bounds for the entries e that have been seen so far. The top-k dominating objects are managed by a min-heap structure W as the algorithm progresses, while g is the current k-th score in W used for pruning. More specifically, the algorithm works as follows:

- First, the root node is loaded, and its entries are inserted into S after upper score bounds have been derived from information in the root node.
- While S contains non-leaf entries, the non-leaf entry e_z with the highest priority is removed from S, the corresponding tree node z is visited, and: i) the dom^u, dom^l scores of existing entries in S that partially dominating e_z are refined using the contents of z, ii) the dom^u, dom^l values for the contents of z are computed and inserted into S.
- W is updated with objects/entries of higher dom^l than g.
- Finally, entries are pruned from S if: i) they cannot lead to objects that may be included in W and ii) they are not partially dominated by entries leading to objects that can reach W.

The Upper-bound Based Traversal Algorithm (UBT) is the PBT algorithm with a priority traversal order guided by the highest upper bound score (dom^u) entries e_z. Such an order would visit the objects that have high probability to be in the top-k dominating results early.

We illustrate the execution of UBT in the example of Figure 4. We denote the score bounds of an entry e by the interval $dom(e) = [dom^l(e), dom^u(e)]$. UBT accesses the root node and its entries are inserted into S after their lower/upper bound scores are derived: $dom(e_1) = [0, 15]$, $dom(e_2) = [0, 9]$, $dom(e_3) = [0, 9]$, $dom(e_4) = [0, 3]$, $dom(e_5) = [0, 3]$. The entry e_1 has the highest dom^u score in S, thus is removed and its child node z is accessed. The score bounds for p_1, p_2, p_4 are computed and the process is repeated until p_2 is reported as the top-1 result.

5) Cost-Based Traversal Algorithm (CBT): A closer look into UBT reveals that the upper score bounds alone may not offer the best priority order for traversing the tree. S can grow very large if there are many partial domination relationships between its entries. To minimize the partially dominating entry pairs in S, the visited nodes are prioritized based on their level at the tree. In addition, between entries at the highest level in S, the one with the highest upper bound is chosen to find the objects with high scores early. This variation of UBT is the Cost-Based Traversal Algorithm (CBT).

B. *Top-k Dominating Algorithms on non-indexed data*

In [7] there is also a study for the evaluation of top-k dominating queries on non-indexed data, assuming that data objects are stored in random order in a disk file.

A practically viable solution is to first bulk-load an aR-tree from the dataset and then compute top-k dominating objects using the previous proposed algorithms. The bulk-loading step requires externally sorting the objects, which is known to scale well for large datasets. However, external sorting may incur multiple I/O passes over data.

In [7] two algorithms, named Coarse-grained Filter Algorithm and Fine-grained Filter Algorithm, are proposed to determine the top-k dominating objects with only a constant number of data passes, by using a filter-refinement framework.

These algorithms require 3 passes over data. The first pass is the counting pass, which employs a memory grid structure to keep track of object count in cells, while scanning over the data. This structure is then used to derive lower/upper bound scores of objects in the next pass. The second pass is the filter pass, which applies pruning rules to discard unqualified objects and keep the remaining ones in a candidate set. The refinement pass, being the final pass, performs a scan over the data to count the exact domination scores of all candidate objects. Eventually, the top-k dominating objects are returned.

C. Progressive Algorithms for Subspace Top-k Dominating Queries

The previous approaches are characterized by several limitations, such as: i) they lack progressiveness (they report the k best objects at the end of the processing), ii) they require a multi-dimensional index or they build a grid-based index on-the-fly, which suffers from performance degradation, especially in high dimensionalities, and iii) they do not support vertically decomposed data. Motivated by these observations, the following proposed algorithms in [12] can handle efficiently any subset of the dimensions in a progressive manner. More specifically, four algorithms have been proposed in [12]: Basic Scan Algorithm (BSA), Union Algorithm (UA), Reverse Algorithm (RA), Differential Algorithm (DA), out of which the best overall performance is shown by DA.

R-trees and their variants have been extensively used in the literature to support a broad range of queries over multi-dimensional data sets, including the top-k dominating queries ([6], [7], [9], [8]). Their major limitation is that since all dimensions are used to organize the dataset, subspace search requires a series of projection operations which affects efficiency due to increased I/O activity. Moreover, the resulting indexing scheme becomes inefficient even for processing queries involving the whole set of dimensions, due to the dimensionality curse.

Alternative approaches have been proposed organizing each dimension separately, resulting in a column-oriented (i.e., vertically decomposed) physical data organization. Column-oriented storage shows significant performance improvements in specific types of operations and moreover, offers a completely independent treatment of dimensions, thus supporting flexible query processing involving a subset of dimensions. This way, the rest of the dimensions do not participate in query processing, and thus, query evaluation becomes more efficient. A column-oriented organization is a natural and intuitive choice taking into account that it is impossible to provide a full-dimensional indexing scheme for every $2^d - 1$ possible subsets of dimensions.

The architecture of the physical organization of the proposed methods in [12] is depicted in Figure 5. Each dimension is organized separately by a B$^+$-tree, which facilitates random as well as sorted access (any one-dimensional access method with random and sorted access support may be used as well). Each B$^+$-tree indexes the attribute values of the specific dimension. A user may select any subset of dimensions, whereas query execution is supported by the use of an LRU buffer. An additional B$^+$-tree is used (instead of another B$^+$-tree a hashing scheme could also be applied), called Set-B$^+$, which is used by the top-k query algorithm for intermediate computations. At the beginning of any top-k query, the Set-B$^+$ is empty. During execution, the IDs of scanned objects are inserted into the Set-B$^+$ and several counters are updated. By using this approach, all intermediate results are kept on disk (if needed) and there is no need of additional main memory storage. The Set-B$^+$ shares the same LRU buffer with all other data B$^+$-trees.

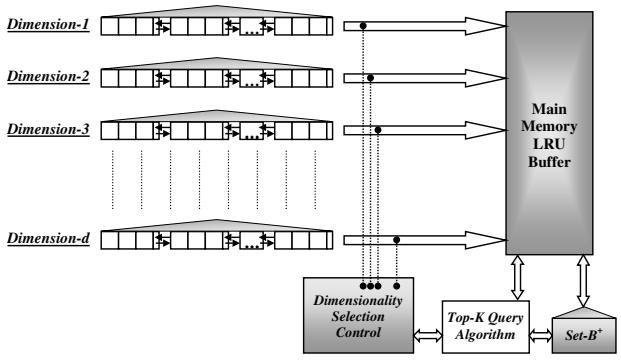

Fig. 5. Data organization utilized by BSA, UA, RA, DA.

The proposed algorithms in [12] are using the concept of terminating objects. A terminating object is an object whose attribute values on all dimensions of interest have been retrieved (scanned) via sorted access on their indexes. The main idea of the proposed algorithms is to scan on B$^+$-trees of the selected dimensions from the beginning in a round-robin manner, to discover the terminating objects one by one, to estimate and calculate (if necessary) their domination scores that are maintained in a max-heap, and to extract them progressively one by one from the max-heap when their scores are definitely greater than the rest objects. The main difference between the proposed algorithms is the way that they compute the exact domination scores of the detected terminating objects.

An example of detection of terminating objects is depicted in Figure 6, after the organization of the dataset of our main example. Two B$^+$-trees are used, one for the x dimension and one for the y dimension. The objects are sorted by their values in each B$^+$-tree. During the round-robin scan, the first detected terminating object is p_2, requiring 4 value accesses. The second detected terminating object is p_4 and requires another 3 value accesses. This process continuous until all relevant objects are reported.

x	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}	p_{11}	p_{12}	p_{13}	p_{14}	p_{15}
	5	11	16	28	33	37	46	50	58	65	72	85	93	103	108
y	p_{13}	p_2	p_4	p_{14}	p_1	p_9	p_6	p_7	p_{12}	p_3	p_{11}	p_{15}	p_8	p_{10}	p_5
	16	22	30	36	45	49	58	65	65	72	74	80	82	95	105

t_1 t_2 t_3

Fig. 6. Example of query execution for BSA, UA, RA, DA.

Due to the sorted order of values in the B$^+$-trees, all objects with the same value in a specific dimension appear sequentially

in the leaf level. This group of objects is called an equality group (the equality group of an object p is denoted E_p). In the example of Figure 6 the objects p_7, p_{12} have the same y value (65), thus they lie in the same equality group ($E_{p_7} = E_{p_{12}} = \{p_7, p_{12}\}$).

The sorted order position of a terminating object p is its position in the dimension that was detected and is denoted as $pos(p)$. In the example of Figure 6 we have $pos(p_2) = 2$, $pos(p_4) = 4$, $pos(p_1) = 5$. The position of any object p in any dimension i is denoted as $Lpos_i(p)$. In the example of Figure 6 we have $Lpos_x(p_1) = 1$, $Lpos_y(p_4) = 3$. In case the object lies in an equality group the left-most position is considered. In the example of Figure 6 we have $Lpos_y(p_{12}) = 8$.

1) Basic Scan Algorithm (BSA): The Basic Scan Algorithm is based on the following fundamental properties:

(a) If p is an object in a dimension with an equality group E_p, then p may be dominated only by objects that are before E_p in the sorted order of that dimension, and it may dominate objects that are after E_p. If there are other objects in E_p, they may dominate p, or they may be dominated by p, or they may not dominate each other.

(b) According to the previous property, if in a dimension an object p has been detected as a terminating object, it cannot dominate more than $N - pos(p) + |E_p| - 1$ objects, where $N = |D|$ the total number of the objects in the dataset D.

This important property allows an immediate estimation of the domination score $estdom(p) = N - pos(p) + |E_p| - 1$ of any terminating object p at the moment it was detected. BSA keeps all discovered terminating objects in a max-heap H prioritized on their estimated domination scores. During processing, only the exact domination values of the top estimated objects are calculated, updating the heap as necessary. The following fundamental property provides the condition for a terminating object to be reported as the next top result, enabling the progressive behavior of the algorithm.

(c) If t_1, t_2 are the top-2 terminating objects in the heap H, and $dom(t_1) \geq estdom(t_2)$ (or $dom(t_1) \geq dom(t_2)$ in case t_2 has an exact score), then t_1 can be immediately reported as the next top result.

Assembling all together, after initializing the heap H, the LRU-Buffer and the Set-B$^+$, the main algorithmic procedure of BSA is the following:

- If H is empty then scan and detect the next two terminating objects, else only the next one, give them their estimated scores and insert them into H.
- Extract the top-1 object t_1 from H and if it has not an exact domination score then calculate it ($dom(t_1)$). Get also the top-2 object t_2 from H (without extraction and calculation).
- If $dom(t_1) \geq estdom(t_2)$ (or $dom(t_1) \geq dom(t_2)$ in case t_2 has an exact score), then report t_1 as the next top dominating result, else go to the first step.
- The process is repeated until all top-k dominating objects are reported.

The exact domination score of a terminating object t in BSA is computed based on the domination check between t and all objects that are in the same equality group with t and after t in the dimension that t was detected. The domination check is performed by retrieving all attribute values of these objects using random accesses. Subsequent algorithms change the way that exact scores are computed toward performance improvement.

Let us present an example of BSA execution by applying a top-1 dominating query to the data depicted in Figure 6. Initially, the first two terminating objects are detected and enheaped, by visiting the objects in the B$^+$-trees for dimensions x and y with the (round-robin) order: p_1, p_{13}, p_2, p_2. Thus, p_2 is the first terminating object (detected after 4 value accesses) and is enheaped with an estimated domination value of $estdom(p_2) = N - pos(p_2) + |E_{p_2}| - 1 = 15 - 2 + 1 - 1 = 13$. Then, the second terminating object p_4 (detected after 3 more value accesses by visiting p_3, p_4, p_4) is enheaped with an estimated domination value of $estdom(p_4) = N - pos(p_4) + |E_{p_4}| - 1 = 15 - 4 + 1 - 1 = 11$. In the next step, p_2 is extracted from the top of the heap (as it has the maximum score), and its exact domination score is computed by checking whether it dominates the following objects $p_4, p_{14}, p_1, p_9, p_6, p_7, p_{12}, p_3, p_{11}, p_{15}, p_8, p_{10}, p_5$, requiring another 13 sequential value accesses in the y dimension and 13 random value accesses in the x dimension (to retrieve both coordinates and check domination). As p_2 does not dominate p_1, its final exact domination score is $dom(p_2) = 12$. It holds that $dom(p_2) = 12 > 11 = estdom(p_4)$, thus the object p_2 can be immediately reported as the top-1 dominating object. BSA requires 4+3+13+13=29 total value accesses.

2) Union Algorithm (UA): Although BSA is progressive, it performs a significant number of random accesses for domination checking, and this may lead to a significant I/O cost. The Union Algorithm (UA) alleviates this problem as it does not require any explicit domination checks among data objects. UA is a variation of BSA. The fundamental difference is that UA uses a different mechanism for exact domination value computation. It takes advantage from an important property (union property) that can calculate the domination score of a terminating object t by using the objects that have already been retrieved before t in the selected dimensions, and not after t (that BSA does).

Union Property: If t is a terminating object, and we collect in a set U_t all objects that lie before the equality groups of t in the selected dimensions, and we collect also in a set UE_t all the equivalent objects to t (i.e., UE_t is the intersection of all the equality groups of t in the selected dimensions), then it holds that: $dom(t) = N - |U_t| - |UE_t| - 1$.

This comes from the fact that t cannot dominate the following: i) objects that have a better value in at least one dimension (i.e., lie before the equality group of t), ii) objects that have the same attribute values in all selected dimensions (equivalent objects), and iii) itself.

Therefore, the exact domination score of a terminating object t in UA is computed based to the union property. The main idea is to compute the sets U_t and UE_t, by scanning the B$^+$-trees again from the beginning in all selected dimensions, and stop when the equality group of t has been reached. It is important to note that during the round-robin scan for the retrieving of t, all scanning pointers stop at $pos(t)$ (or

$pos(t) - 1$), thus to detect the $Lpos$ positions of the equality groups of t and at the same time to compute the sets U_t and UE_t, we scan again from the beginning each dimension separately. It is also important to note that the required sets U_t, UE_t are not implemented as separate structures and no additional space is required. This is because we are not interested in the specific final objects that are contained, but only in their size (e.g., $|U_t|, |UE_t|$), and this is conducted with specific counters and counting in the Set-B$^+$.

In our running example that was used also for BSA (Figure 6), the only difference is that when p_2 is extracted from the top of the heap, the calculation phase of the UA algorithm computes the size of its union set $|U_{p_2}| = |\{p_1, p_{13}\}| = 2$ and of the set of its equivalent objects $|UE_{p_2}|$=0, by making 4 total value accesses from the beginning. Therefore, the domination score of p_2 is calculated as follows: $dom(p_2) = 15 - 2 - 0 - 1 = 12$. Now the algorithm terminates performing in total 4+3+4=11 value accesses (less and not random).

3) Reverse Algorithm (RA): Although UA eliminates the drawbacks of BSA, it has a serious limitation: when the extracted terminating objects have high positions (especially in anti-correlated data), or there are a lot of selected dimensions, the total number of required value accesses in set calculations is significantly increased and this may produce additional I/O cost. The Reverse Algorithm (RA), takes advantage of the fact that all positions of the detected terminating object t in the selected dimensions are less than or equal to $pos(t)$. Therefore, instead of scanning the B$^+$-trees from the beginning (as UA does), RA scans backwards from $pos(t)$ and stops when the equality group of t has been retrieved completely in each selected dimension (i.e., its $Lpos$ positions reached). Initially, the set U_t contains all objects found so far during the round-robin scan to the position $pos(t)$, and during the reverse scanning, all detected objects are definitely lying after (or inside) the equality groups of t, thus they removed from U_t. Therefore, when the reverse scanning is completed U_t will have the appropriate number of objects. It is important to note again that we are not interested in the specific final objects that are contained in $|U_t|, |UE_t|$, but only in their size (e.g., $|U_t|, |UE_t|$), and this is conducted with specific counters and counting in the Set-B$^+$.

In our running example (Figure 6), the only difference is that when p_2 is extracted from the top of the heap, the calculation phase of the RA algorithm computes the size of its union set $|U_{p_2}| = |\{p_1, p_{13}\}| = 2$ and of the set of its equivalent objects $|UE_{p_2}| = 0$, by making only 2 value accesses from $pos(p_2) = 2$ backwards. Therefore, the domination score of p_2 is calculated as follows: $dom(p_2) = 15 - 2 - 0 - 1 = 12$. Now the algorithm terminates performing in total 4+3+2=9 value accesses.

4) Differential Algorithm (DA): The Differential Algorithm (DA) is based on the idea that when there is a need to compute the exact domination score of a terminating object t, we can select a previously determined convenient terminating object t_p (best object) whose exact domination score has been already computed. Forward and backward scans are performed taking into consideration the position of t_p. Therefore, if object t is positioned after t_p in a selected dimension, then the algorithm scans forward from t_p to t in the B$^+$-tree of this dimension and increases specific counters into the Set-B$^+$ structure for the visited objects, otherwise it scans backwards and decreases specific counters into the Set-B$^+$ structure for the visited objects. Finally, the exact domination value of the terminating object t is computed using the updated visits set counters and the same formula as in UA. The positions of t and t_p that are checked are their $Lpos$ positions which are also stored in the Set-B$^+$ structure during processing.

For example let us considered that the algorithm has already computed the exact domination score of p_4 ($dom(p_4) = 10$) and has been set it as the current best object (Figure 6). To compute the exact score of terminating object p_1, we initially set $dom(p_1) = dom(p_4) = 10$ and we scan backwards in dimension x (as $Lpos_x(p_1) < Lpos_x(p_4)$) till p_1 is found. During scan only one object (p_3) was found exclusively on x dimension, thus we subtract 1 from the score ($dom(p_1) = 9$). Now we scan forward in dimension y (as $Lpos_y(p_1) > Lpos_y(p_4)$) till p_1 is found. During scan only one object (p_{14}) was newly found on y dimension, thus we add 1 to the score ($dom(p_1) = 10$). Thus the final exact domination score of p_1 is $dom(p_1) = 10$.

The selection of best terminating object t_p during processing is based on the minimum absolute difference of positions. Therefore, when a newly discovered terminating object t minimizes the absolute difference of $Lpos$ positions with the current best object t_p, then it becomes the new best object. Using this technique the total number of value accesses is further decreasing.

To further decrease the number of value accesses and the corresponding I/O operations, a collection of pruning rules has been designed toward performance boost, which can be applied in BSA, UA, RA, DA. Three types of pruning rules are proposed in [12], according to their applicability to the algorithm parts: i) Discard Rules (DRs), which update all objects that can be discarded during the scanning for the next candidate terminating object, ii) Early Pruning Rules (EPRs), which are applied before the computation of the exact domination value of the selected terminating object, and iii) Internal Pruning Rules (IPRs), which are applied during exact score computation. For further details see [12].

IV. Metric-Based Top-k Dominating Algorithms

All previous algorithms address the problem in settings where data objects are multi-dimensional objects. 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 domination relation is still meaningful and valid. In [14] there is a first study for processing top-k dominating queries over distance-based dynamic attribute vectors, defined over a metric space. Four progressive algorithms are proposed: Skyline-Based Algorithm (SBA), Aggregation-Based Algorithm (ABA), Pruning-Based Algorithms (PBA1) and (PBA2), from which PBA2 shows the best overall performance.

Let D be a dataset in which a distance function $d()$ has been defined which quantifies the dissimilarity between data objects in D (D is a metric space). Let also 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, if and only if p has an equal or smaller distance than r to all query objects $q_i \in Q$, and p has a smaller distance than r to at least one query object. In case the objects p and r have exactly the same distance from q_i, i.e., $d(p, q_i) = d(r, q_i), \forall i = 1, \ldots, m$, they are called equivalent. The set of objects in D which are not dominated by any other object (according to the distances from Q) is called the metric space skyline with respect to Q, denoted as $MSS(Q)$. A Metric-Based Top-k Dominating Query returns the k objects with the maximum domination scores dom in D respecting the previous defined domination relationship, and is denoted as $MSD(Q, k)$.

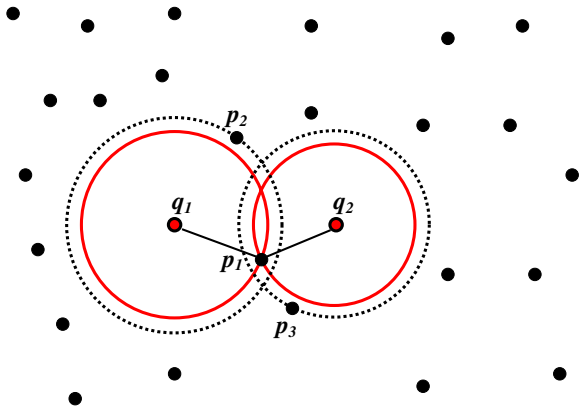

Fig. 7. Example of a metric-based top-3 dominating query.

Figure 7 shows a top-3 dominating query in a metric space with 25 2-dimensional objects, under the Euclidean distance. Two query objects q_1 and q_2 are also depicted. Object p_1 is definitely the top-1 object, as it is the nearest neighbor of both query objects q_1, q_2. Since no other object lies inside either circle $C_1(q_1, d(q_1, p_1))$ or circle $C_2(q_2, d(q_2, p_1))$, p_1 dominates all other objects and its domination score is $dom(p_1) = 24$. The second nearest neighbor of q_1 is p_2, whereas that of q_2 is p_3. Since $d(p_2, q_1) < d(p_3, q_1)$ (p_3 lies outside circle $C_3(q_1, d(p_2, q_1))$) and $d(p_2, q_2) > d(p_3, q_2)$ (p_2 lies outside circle $C_4(q_2, d(p_3, q_2))$), p_2 and p_3 do not dominate each other. However, p_2 and p_3 dominate all other objects since there are no objects lying inside the corresponding dotted circles (except p_1). Thus their domination score is $dom(p_2) = 22$ and $dom(p_3) = 22$, whereas remaining objects have a domination score less than 22. Therefore, the set $\{p_1, p_2, p_3\}$ is the final answer to the top-3 dominating query based on query objects q_1 and q_2.

The data organization used by the algorithms in [14] is depicted in Figure 8. Among the available metric-based indexes the M-tree [1] has been selected 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, the proposed methods are orthogonal to the indexing scheme used, as long as incremental k-nearest-neighbor queries are supported. 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 (similar with the Set-B$^+$). Both the M-tree and

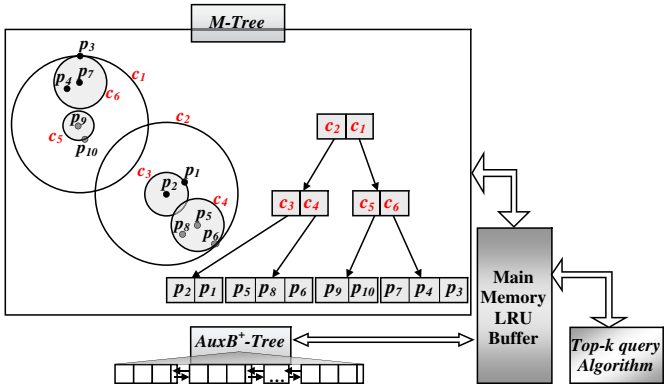

Fig. 8. Data organization used by metric-based top-k dominating algorithms.

the AuxB^+-tree are supported by an LRU buffer which reduces I/O cost due to locality of references.

A. Skyline-Based Algorithm (SBA)

The Skyline-Based Algorithm (SBA) is based directly on the observation of [9] that the top-1 dominating object belongs to the skyline. The process of SBA is the following:

- The skyline S of D with respect to Q is computed ($MSS(Q)$).
- The top-1 dominating object p in D from S is computed, by computing the domination scores of all objects in S.
- p is reported and removed from the dataset D, and the process is repeated until all top-k objects have been reported. Then we re-insert them to D.

For the metric-space skyline query $MSS(Q)$, the state-of-the-art algorithm B^2MS^2 proposed in [2] is used, implemented to manage M-tree nodes.

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

B. Aggregation-Based Algorithm (ABA)

The Aggregation-Based Algorithm (ABA) takes advantages of the properties of the sum-aggregate nearest-neighbor queries studied in [10]. A sum-aggregate nearest neighbor query, denoted as $ANN(Q, k)$, contains the k objects with the minimum aggregate distance (computed based on the sum of distances from Q). An important property is that the first sum-aggregate nearest-neighbor of Q is always a metric space skyline object, i.e., $ANN(Q, 1) \in MSS(Q)$. Therefore, the skyline computation is not required and this enables an alternative methodology which is the main process of the ABA algorithm:

- Initially, the top-1 sum-aggregate nearest-neighbor object p of Q is computed, i.e., $p = ANN(Q, 1)$.

Since p is a skyline object, there is no object that dominates p. Therefore, there are no objects inside p's dominator region and additionally p dominates all other objects lying into its domination region. This ensures that the top-1 dominating object cannot lie into the dominator/domination regions of p.

- Thus, the top-1 dominating object is searched in the rest of the dataset (remaining region C). Candidates 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.
- The dominating scores of all objects in C are computed and the top-1 dominating object t is retrieved.
- t is reported and removed from the dataset D, and the process is repeated until all top-k objects have been reported. Then we re-insert them to D.

For the aggregate nearest neighbor query $ANN(Q, 1)$ we use the MBM algorithm of [10] 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. The range queries are efficiently supported by the M-tree structure. The candidate objects of the set C and their domination scores are kept and updated into the AuxB$^+$-tree.

For a better view of those candidates let us consider the example of Figure 9 for a Top-1 dominating query with two query objects q_1, q_2. After $p = ANN(Q, 1)$ is retrieved, the candidates are the objects contained inside the circles with centers the query objects and radii their corresponding distances from p (shaded area). In our case, there are no objects inside the circle intersection area R_{in} (dominator region of p), 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. Additionally, p dominates all objects outside the two circles R_{out} (domination region of p).

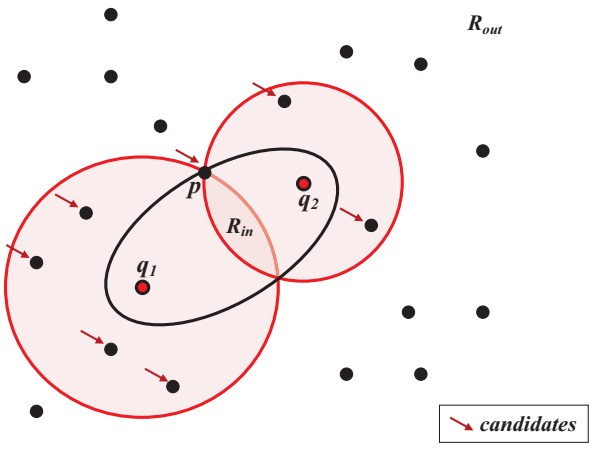

Fig. 9. Example for ABA algorithm.

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 plus the cost of the simple range queries, is lower than the cost of a complete skyline computation (as performed by SBA). The limitations of ABA are as follows: i) it recalculates up to k times the domination scores of some nearest neighbor candidate objects of C, ii) when the cardinality of Q increases we must perform a large number of range queries, which deteriorates the performance of the algorithm, and when the query objects are far from each other, the range queries may return a large number of candidates, thus C grows significantly leading to high computational costs.

C. Pruning-Based Algorithms

Based on the significant limitations of SBA and ABA, we focus into a third alternative. The key idea behind the Pruning-Based Algorithms (PBA1, PBA2) is to incrementally retrieve the nearest neighbors of the query objects in a round-robin fashion, to compute the domination scores of common neighbors, and, under certain conditions, to extract the top-k results in a progressive manner. The algorithmic process of PBA1 is similar to the RA algorithm which already described. The difference is that instead of scanning the B$^+$-trees values (since we don't have multiple B$^+$-trees but a single M-tree), the retrieving of the nearest neighbors from each query object is done dynamically by an incremental nearest neighbor retrieval process which is efficiently supported by the M-tree implementation of [1].

Now the notion of a terminating object in the metric space is equivalent to the notion of a "common neighbor object", which is an object that has been discovered in the nearest neighbor order lists of all query objects during the incremental NN retrieval (not necessarily with the same position order).

Again we have an upper bound for the domination score of a retrieved object o_i, which is used as an estimation of its score denoted as $estdom(o_i)$. This bound is based on the property: If an object o_i has been retrieved as the $(r_{i,j})$-th nearest-neighbor of query object q_j (where $r_{i,j}$ is the rank position of o_i in the nearest neighbor order list of q_j), then: $dom(o_i) \leq N - max_j(r_{i,j}) + eq(o_i)$, where $eq(o_i)$ is the number of all equivalent objects to o_i.

Therefore, after initializing the heap H, the LRU-Buffer and the AuxB$^+$-tree, the main algorithmic process of both PBA1, PBA2 is the following:

- If H is empty then retrieve the next two common neighbor objects (by using a round-robin incremental nearest neighbor retrieval from the query objects), else only the next one, give them their estimated scores and insert them into H.
- Extract the top-1 object t_1 from H and if it has not an exact domination score then calculate it ($dom(t_1)$). Get also the top-2 object t_2 from H (without extraction and calculation).
- If $dom(t_1) \geq estdom(t_2)$ (or $dom(t_1) \geq dom(t_2)$ in case t_2 has an exact score), then report t_1 as the next top dominating result, else go to the first step.
- The process is repeated until all top-k dominating objects are reported.

For the exact domination score computation again the union property is used, thus: If t is a common neighbor object, and we count in $|U_t|$ all objects that have distances strictly smaller than t in the selected dimensions, and we count also in $|UE_t|$ all the equivalent objects to t, then it holds that: $dom(t) = N - |U_t| - |UE_t| - 1$.

To compute $|U_t|$ and $|UE_t|$, the algorithm PBA1 (similar with RA) scans backwards into the nearest neighbor list of each query object, until the object t is detected. But PBA2 goes one step further. As all these objects are already retrieved and inserted into the AuxB$^+$-tree, the required counting can be successfully performed inside the AuxB$^+$-tree by using some additional counters, without materializing the sets U_t, UE_t and without scanning again the nearest neighbor lists.

The methodology of algorithms PBA1 and PBA2 enable the usage of several pruning heuristics, which reduce the runtime costs further. Three different types of pruning heuristics have been proposed in [14]: i) Discard Heuristics - DH, which can discard objects that have not been retrieved yet, ii) Early Pruning Heuristics - EPH, which can prune objects before the calculation of their exact domination scores, and iii) an Internal Pruning Heuristic - IPH, which can prune objects during the procedure of the exact domination score calculation. For further details see [14].

V. Variations of Top-k Dominating Queries

In [16] there is a study of the problem of computing top-k dominating queries efficiently on uncertain data. A threshold-based probabilistic top-k dominating algorithm is proposed with an accuracy guarantee.

In [4] there is a first study of top-k dominating query processing algorithms in a streaming environment. Three exact algorithms (BFA, EVA, ADA) and two approximate algorithms that trade accuracy for speed (AHBA and AMSA) are proposed. AHBA offers probabilistic guarantees regarding the accuracy of the result based on the Hoeffding bound, whereas AMSA performs a more aggressive computation resulting in more efficient processing.

In [15] there is a study of how to rank spatial objects with respect to their non-spatial attributes within their spatial neighborhoods. To enable a general ranking, a ranking function that inherits the advantages of domination relationship and integrates them with spatial proximity is used. The result is a top-k neighborhood domination query. An effective index structure, and a branch and bound algorithm that executes the ranking efficiently via the index is proposed.

VI. Conclusions

Top-k dominating queries combine the advantages of regular top-k and skyline queries, by bounding the size of the result without the need for user-defined scoring functions. In the last years, they became an active research area due to their importance in several modern applications. In this paper, we presented efficient algorithms that answering top-k dominating queries: i) in general and multi-dimensional data, ii) in indexed and non-indexed data, iii) in data that lying into a metric space and in dynamic environments. There is also a growing interest for studying top-k dominating queries in streaming environments, in uncertain data and other variations.

An interesting and challenging direction for future research is to enable domination-based preference queries in distributed computing platforms such as Hadoop and Spark.

References

[1] P. Ciaccia, M. Patella and P. Zezula: "M-tree: An Efficient Access Method for Similarity Search in Metric Spaces", *Proceedings 23rd International Conference on Very Large Data Bases (VLDB)*, pp.426-435, Athens, Greece, 1997.

[2] D. Fuhry, R. Jin and D. Zhang: "Efficient Skyline Computation in Metric Space", *Proceedings 12th International Conference on Extending Database Technology (EDBT)*, pp.1042-1051, Saint Petersburg, Russia, 2009.

[3] A. Guttman: "R-Trees, A Dynamic Index Structure for Spatial Searching", *Proceedings ACM International Conference on Management of Data (SIGMOD)*, pp.47-57, Boston, MA, 1984.

[4] M. Kontaki, A. N. Papadopoulos and Y. Manolopoulos: "Continuous Top-k Dominating Queries", *IEEE Transactions on Knowledge and Data Engineering*, Vol.24, No.5, pp.840-853, 2012.

[5] I. Lazaridis and S. Mehrotra: "Progressive Approximate Aggregate Queries with a Multi-Resolution Tree Structure", *Proceedings ACM International Conference on Management of Data (SIGMOD)*, pp.401-412, Santa Barbara, CA, 2001.

[6] M. L. Yiu and N. Mamoulis: "Efficient Processing of Top-k Dominating Queries on Multi-Dimensional Data", *Proceedings 33rd International Conference on Very Large Data Bases (VLDB)*, pp.483-494, Vienna, Austria, 2007.

[7] M. L. Yiu and N. Mamoulis: "Multi-dimensional top-k dominating queries", *The VLDB Journal*, Vol.18, No.3, pp.695718, 2009.

[8] D. Papadias, P. Kalnis, J. Zhang and Y. Tao: "Efficient OLAP Operations in Spatial Data Warehouses", *Proceedings of the 7th International Symposium on Advances in Spatial and Temporal Databases (SSTD)*, pp.443-459, Redondo Beach, CA, 2001.

[9] D. Papadias, Y. Tao, G. Fu and B. Seeger: "Progressive Skyline Computation in Database Systems", *ACM Transactions on Database Systems*, Vol.30, No.1, pp.4182, 2005.

[10] D. Papadias, Y. Tao, K. Mouratidis and C. K. Hui: "Aggregate Nearest Neighbor Queries in Spatial Databases", *ACM Transactions on Database Systems*, Vol.30, No.2, pp.529-576, 2005.

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

[12] E. Tiakas, A. N. Papadopoulos and Y. Manolopoulos: "Progressive Processing of Subspace Dominating Queries", *The VLDB Journal*, Vol.20, No.6, pp.921-948, 2011.

[13] E. Tiakas, A. N. Papadopoulos and Y. Manolopoulos: "On Estimating the Maximum Domination Value and the Skyline Cardinality of Multi-Dimensional Data Sets", *International Journal of Knowledge-Based Organizations*, Vol.3, No.4, pp.61-83, 2013.

[14] E. Tiakas, G. Valkanas, A. N. Papadopoulos, Y. Manolopoulos and D. Gunopoulos: "Metric-Based Top-k Dominating Queries", *Proceedings of the 17th International Conference on Extending Database Technology (EDBT)*, pp.415-426, Athens, Greece, 2014.

[15] X. Xie, H. Lu, J. Chen and S. Shang: "Top-k Neighborhood Dominating Query", *Proceedings of the 18th International Conference on Database Systems for Advanced Applications (DASFAA)*, pp.131-145, Wuhan, China, 2013.

[16] W. Zhang, X. Lin, Y. Zhang, J. Pei and W. Wang: "Threshold-based Probabilistic Top-k Dominating Queries", *The VLDB Journal*, Vol.19, No.2, pp.283-305, 2010.