Efficient Index-Based Approaches for Skyline Queries in Location-Based Applications

Ken C.K. Lee, Baihua Zheng, *Member*, *IEEE*, Cindy Chen, and Chi-Yin Chow, *Member*, *IEEE*

Abstract—Enriching many location-based applications, various new skyline queries are proposed and formulated based on the notion of *locational dominance*, which extends conventional one by taking objects' nearness to query positions into account additional to objects' nonspatial attributes. To answer a representative class of skyline queries for location-based applications efficiently, this paper presents two index-based approaches, namely, *augmented R-tree* and *dominance diagram*. Augmented R-tree extends R-tree by including aggregated nonspatial attributes in index nodes to enable dominance checks during index traversal. Dominance diagram is a solution-based approach, by which each object is associated with a precomputed *nondominance scope* wherein query points should have the corresponding object not locationally dominated by any other. Dominance diagram enables skyline queries to be evaluated via parallel and independent comparisons between nondominance scopes and query points, providing very high search efficiency. The performance of these two approaches is evaluated via empirical studies, in comparison with other possible approaches.

Index Terms—Locational dominance, skyline query, reverse skyline query, subspace skyline query, top-K query, index, search algorithms, performance

1 Introduction

Location-dependent spatial queries, which find spatial objects based on their nearness to given query points, have been extensively studied. Nearest neighbor (NN) search [9], [23], reverse NN search [12], and so on, are representative ones. On top of those spatial queries, nonspatial attributes of objects may be considered together as additional yet independent query criteria. For instance, a query "finding the nearest hotel with price below $250" uses hotel price, i.e., a nonspatial attribute, to distill inexpensive hotels on top of NN search. Another query "finding the cheapest hotel within a mile" finds the cheapest hotel within a range query result.

In fact, many today's applications would consider both object nearness and their nonspatial attributes as object selection and ranking criteria, simultaneously. Thus, many recent research works, for example, [10], [11], [14], [17], [27], [29] have explored the notion of *locational dominance* in location-dependent information access.

Formally, we consider a set of spatial objects $\mathcal{O}$ distributed in a 2D geographical space $\mathcal{S}$ in this paper. Each object $o(\in \mathcal{O})$ is located at a point location (i.e., an (x, y) coordinate) bounded by $\mathcal{S}$ and has values for a set of nonspatial attributes, $\mathcal{A}$. Notationally, we use $|o, p|$ to denote the euclidean distance between a location point p (within $\mathcal{S}$) and o; and we use $o[a]$ to denote o's value for an attribute $a(\in \mathcal{A})$. Here, all attribute values are assumed to be ordinal and a smaller attribute value is considered to be better. Accordingly, Definition 1 gives the definition of locational dominance; and Example 1 as the running example illustrates it.

Definition 1 (Locational Dominance). *With respect to a query point q, we denote that an object o locationally dominates another object o' by $o \vdash_q o'$, which is expressed as follows:*

$$o \vdash_q o' = \forall_{a\in\mathcal{A}}\, o[a] \le o'[a] \wedge |o, q| \le |o', q| \wedge (\exists_{a\in\mathcal{A}} o[a] < o'[a] \vee |o, q| < |o', q|). \quad (1)$$

Besides, we use $o \nvdash_q o'$ to indicate that o does not locationally dominate o' with respect to q.

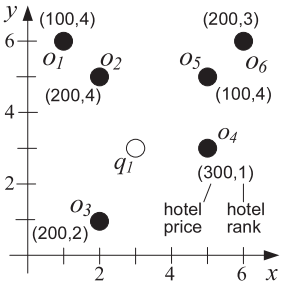
Example 1 (Running Example). Fig. 1 depicts the locations of six example hotels (objects), namely, o_1, o_2, o_3, o_4, o_5, and o_6, along with their prices and ranks, which are nonspatial attributes. The smaller its price (rank) is, the better a hotel is considered to be. The euclidean distances of objects to a point of attraction, q_1, (i.e., a query point) are listed in the figure.

With respect to q_1, o_2, and o_6, which cannot offer a lower price, a better rank, or a shorter distance to q_1 than o_3, are said to be locationally dominated by o_3. Likewise, o_1 is locationally dominated by o_5, since o_5 is closer to q_1 than o_1, despite they have the same price and rank. Because of its best rank, o_4 is not dominated. Thus, o_3, o_4, and o_5 are those not locationally dominated objects with respect to q_1.

Based on this locational dominance notion, many new spatial queries can be devised to enrich location-based applications. In this paper, we consider four representative ones, namely,

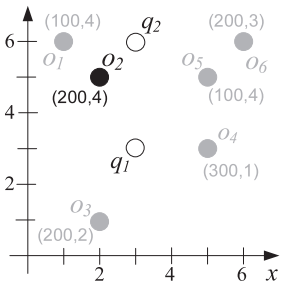
- *K.C.K. Lee is with Amazon.com. E-mail: kcklee@amazon.com.*
- *B. Zheng is with the School of Information Systems, Singapore Management University, 80 Stamford Rd, Singapore 178902. E-mail: bhzheng@smu.edu.sg.*
- *C. Chen is with the Department of Computer Science, University of Massachusetts Lowell, One University Avenue, Lowell, MA 01854. E-mail: cchen@cs.uml.edu.*
- *C.-Y. Chow is with the Department of Computer Science, City University of Hong Kong, 83 Tat Chee Avenue, Kowloon, Hong Kong. E-mail: chiychow@cityu.edu.hk.*

Manuscript received 5 June 2011; revised 9 Jan. 2012; accepted 21 Sept. 2012; published online 26 Oct. 2012.
Recommended for acceptance by C. Shahabi.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-2011-06-0325.
Digital Object Identifier no. 10.1109/TKDE.2012.216.



	distance to q_1
o_1	$\sqrt{13}$
o_2	$\sqrt{5}$
o_3	$\sqrt{5}$
o_4	2
o_5	$\sqrt{8}$
o_6	$\sqrt{18}$

Fig. 1. Six example objects and an SQ issued at q_1.



	q_1	q_2
o_1	$\sqrt{13}$	2
o_2	$\sqrt{5}$	$\sqrt{2}$
o_3	$\sqrt{5}$	$\sqrt{29}$
o_4	$\sqrt{2}$	$\sqrt{13}$
o_5	$\sqrt{8}$	$\sqrt{5}$
o_6	$\sqrt{18}$	3

Fig. 2. RSQ issued at object o_2.

1. location-dependent skyline query (SQ),
2. reverse location-dependent skyline query (RSQ),
3. top-K influential object query (TIQ), and
4. subspace location-dependent skyline query (SSQ), and develop holistic approaches to answer them efficiently.

Those queries are detailed as follows:

- *Location-dependent skyline query (SQ).* Given a query point q and an object set $\mathcal{O}$, an SQ (which has also been studied in [10], [11], [14], [17], [27], [29]) finds all objects that are not locationally dominated by others with respect to q, formally, $\mathrm{SQ}(q, \mathcal{O}) = \{o \mid o \in \mathcal{O}, \nexists_{o' \in \mathcal{O}-\{o\}} o' \vdash_q o\}$. In the running example, an SQ issued at q_1 returns $\{o_3, o_4, o_5\}$.
- *Reverse location-dependent skyline query (RSQ).* As opposed to SQ, an RSQ (that borrows the idea of reverse skyline query [7]) searches for query points (from a given query point set $\mathcal{Q}$) with respect to which a specified object o is not locationally dominated. Hence, $\mathrm{RSQ}(\mathcal{Q}, o) = \{q \mid q \in \mathcal{Q}, \nexists_{o' \in \mathcal{O}-\{o\}} o' \vdash_q o\}$, or $\mathrm{RSQ}(\mathcal{Q}, o) = \{q \mid q \in \mathcal{Q}, o \in \mathrm{SQ}(q, \mathcal{O})\}$.[1]

  As shown in Fig. 2 that considers an additional point of attraction q_2 (that means $\mathcal{Q} = \{q_1, q_2\}$), an RSQ issued at o_2 retrieves q_2 (not q_1), since o_2 is locationally dominated by o_3 with respect to q_1. This RSQ is especially useful for a hotel (e.g., o_2) to identify customers (interested in q_2) who may prefer it to other hotels.
- *Top-K influential object query (TIQ).* SQ can return those not locationally dominated objects, but they cannot rank those nondominated objects. In many situations, it is more desirable if there is a way to tell which objects are valuable to be recommended given multiple query points. Accordingly, we introduce and define the degree of influence for an object o in $\mathcal{O}$ denoted by $I_o(\mathcal{Q})$ as the number of SQ results (each for one query point in a given query point set $\mathcal{Q}$) that contain o to weigh the importance of the object. More formally, $I_o(\mathcal{Q}) = |\{q \mid q \in \mathcal{Q}, o \in \mathrm{SQ}(q, \mathcal{O})\}|$, or $|\mathrm{RSQ}(\mathcal{Q}, o)|$. Let us revisit Fig. 2. With respect to $\{q_1, q_2\}$, $I_{o_4}(\{q_1, q_2\})$ and $I_{o_2}(\{q_1, q_2\})$ are both 2, whereas all others are 1. That means o_2 and o_4 are not locationally dominated with respect to the two points of attraction. Hence, these hotels are said to be more influential and more valuable to be recommended to customers.

  Here, we introduce TIQ to retrieve K objects that have the highest degrees of influence to $\mathcal{Q}$; notationally $\mathrm{TIQ}(\mathcal{Q}, \mathcal{O}) = \{o' \mid o' \in \mathcal{O}' \subseteq \mathcal{O} \wedge |\mathcal{O}'| = K, \forall_{o \in (\mathcal{O}-\mathcal{O}')} I_o(\mathcal{Q}) \leq I_{o'}(\mathcal{Q})\}$[2,3,4].
- *Subspace location-dependent skyline query (SSQ).* An SSQ considers a *proper subset* of nonspatial attributes ($\mathcal{A}' \subset \mathcal{A}$) and object proximities to a query point q, and it retrieves those nonlocationally dominated objects according to *subspace locational dominance* as defined in Definition 2.

Definition 2 (Subspace Locational Dominance). *With respect to a query point q, we denote that an object o subspace locationally dominates another object o' by $o \vdash_{q,\mathcal{A}} o'$, as expressed in the following equation:*

$$o \vdash_{q,\mathcal{A}'} o' = \forall_{a \in \mathcal{A}'}\, o[a] \leq o'[a] \wedge |o, q| \leq |o', q| \\ \wedge (\exists_{a \in \mathcal{A}'}\, o[a] < o'[a] \vee |o, q| < |o', q|), \quad (2)$$

where $\mathcal{A}' \subset \mathcal{A}$. We also use $o \nvdash_{q,\mathcal{A}'} o'$ to denote that o does not subspace locationally dominate o' according to q and $\mathcal{A}'$.

Thus, $\mathrm{SSQ}(q, \mathcal{O}, \mathcal{A}') = \{o \mid o \in \mathcal{O}, \nexists_{o' \in \mathcal{O}-\{o\}} o' \vdash_{q,\mathcal{A}'} o\}$. In Fig. 1, when only hotel price is considered, an SSQ at q_1 retrieves o_2, o_3, o_4, and o_5. Now, o_2 becomes included compared with SQ at q_1, since both o_1 and o_2 are equidistant to q_1 and have the same good hotel price.

Only at search time can object distances to query points and so locational dominance relationships among objects be determined. Hence, the aforementioned skyline queries cannot trivially be answered through conventional index-based skyline approaches that require all fixed object attribute values. Nonindexed approaches inevitably access all objects to determine their distances to query points and incur exhaustive object comparisons for *dominance checks*, leading to poor search performance.

In this paper, we develop two index-based approaches, namely, 1) *augmented R-tree* (AR-Tree) and 2) *dominance diagram* (DD), as holistic solutions to efficiently answer the aforementioned skyline queries efficiently. AR-tree extends R-tree [26] that indexes objects on their locations by aggregating objects' nonspatial attribute values and maintaining aggregated attribute values in index nodes for search space pruning. Searches for nonlocationally

1. The differences between RSQ and [7] are discussed in Section 2.

2. Here, K is bounded by the size of an object set $|\mathcal{O}|$.

3. If multiple objects have identical degree of influence, the tie is resolved arbitrarily.

4. The work [22] defines the influence for an object as the number of top-K query results that include the object, but differently, we consider the quantity of SQ results instead.

dominated objects with AR-tree resemble iterative NN searches [9] and use the aggregated nonspatial attributes to determine the presence of nonlocationally dominated objects within certain parts of an index.

DD is a solution-based approach. It maintains a collection of *nondominance scopes*. Each nondominance scope is a spatial area for an object $o(\in \mathcal{O})$ denoted by $\mathcal{D}(o)$, such that o is guaranteed to be not locationally dominated by any other with respect to any query point inside $\mathcal{D}(o)$. By checking the residence of a query point q in $\mathcal{D}(o)$, whether or not o is locationally dominated with respect to q can be quickly determined. Since locational dominance relationship among objects with respect to all possible locations in $\mathcal{S}$ is precomputed, object comparisons for dominance checks are completely waived at search time. The notion of nondominance scope was first introduced in our previous work [14] to answer location-dependent skyline queries. Here, we extend the idea to support other queries and explore parallel processing to boost search efficiency. We also deal with nondominance scope maintenance in presence of object set updates.

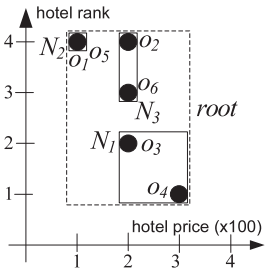
We conducted empirical studies to evaluate the performance of our proposed approaches in comparison with the state-of-the-art approaches designed for conventional skyline queries. DD is the most efficient approach for SQ, RSQ, and TIQ, whereas AR-tree outperforms DD for SSQ.

The remainder of the paper is organized as follows: Section 2 reviews related research works. Sections 3 and 4 detail augmented R-tree and dominance diagram, respectively. Further, Section 5 discusses performance evaluation. At last, Section 6 concludes this paper, and states our contributions and future work directions.

2 Related Work

Since the introduction of skyline query [4] to retrieve nondominated records (a.k.a. maximum/minimum vectors [2] or Pareto-optimal tuples [6]) from a database, various search techniques have been proposed; and they are categorized into sort-based approaches (e.g., SFS [6], SaLSa [1]) and index-based approaches (e.g., Index [25], NN [13], BBS [21], Z-SKY [15], ZINC [19]).

Index-based approaches generally outperform sort-based approaches when dominated objects are well grouped together for search space pruning. Here, we review BBS [21], one of the most well received index-based approaches in the following to explain this point. As shown in Fig. 3, our six hotel objects are indexed on price and rank and bounded as minimum bounding boxes (MBBs) in an R-tree. To find nondominated hotels, BBS accesses unexamined objects and index nodes according to their smallest sums of prices and ranks. The trace of BBS is depicted in the figure. After the root is visited, N_1 is expanded. Then, its objects o_3 and o_4 are examined in sequence. Since they are not dominated, o_3 and o_4 are both collected in a skyline query result set. Next, N_3 is explored. However, it is dominated by o_3 and thus all its contained objects are waived from further examination. Further, N_2 is explored; and o_1 and o_5 are not dominated and collected. Finally, the skyline query result set, $\{o_1, o_3, o_4, o_5\}$, is obtained.



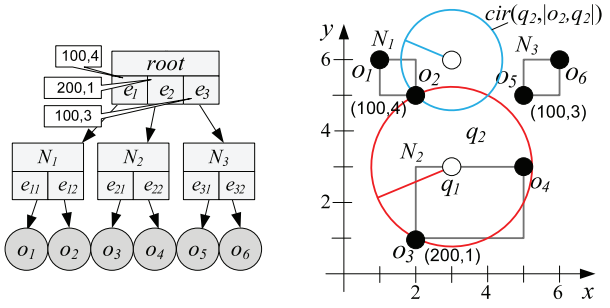
skyline	exam.	pending
$\emptyset$	root	-
$\emptyset$	N_1	N_3, N_2
$\emptyset$	o_3	o_4, N_3, N_2
$\{o_3\}$	o_4	N_3, N_2
$\{o_3, o_4\}$	N_3	N_2
$\{o_3, o_4\}$	N_2	-
$\{o_3, o_4\}$	o_1	o_5
$\{o_1, o_3, o_4\}$	o_5	-
$\{o_1, o_3, o_4, o_5\}$		-

Fig. 3. BBS.

Two possible extensions can enable BBS to answer SQs (SSQs), as discussed below. The first extension indexes both object locations and nonspatial attributes in R-tree. Due to curse of dimensionality [3], the access performance of R-tree, however, deteriorates. The second extension puts the bounds of object locations in index nodes. Accordingly, the accesses of objects and index nodes are ordered based primarily on their smallest sums of nonspatial attributes. In case of tie, one with the shortest euclidean distance to a query point is examined first. Thus, BBS runs with almost the same trace for a query point q_1 except that N_3 is explored (since it is not locationally dominated by o_3) and o_5 is accessed before o_1. Yet, the range of nonspatial attribute values is often narrower than that of spatial coordinates. Far apart objects would be grouped into some index nodes. Thus, many false hits and larger computational and I/O costs can be incurred.

Since BBS and the two possible extensions cannot perform well for SQs (SSQs), sort-based approaches appear to be a good alternative. SaLSa [1] is the most representative sort-based approach. It examines objects according to their minimum attribute values and uses the smallest of the maximum examined objects' attribute values to shrink the result search bound. The search terminates as long as no unexamined objects stay in the search bound. Recently, two other sort-based approaches [11], [29] for SQs have been devised. Since objects close to a query point q should not be locationally dominated by those far away objects, in [11], objects are scanned in an ascending order of their distances to q and those not locationally dominated are maintained as the query result. In [29], δ-scanning approach presumes that conventional skyline SL (based on nonspatial attributes) is already determined, since it is independent of q. Then it filters out those locationally dominated objects from SL. Next, it accesses objects (not in SL) in distance order with respect to q and collects those not locationally dominated and those in SL into an SQ result. Differently, our approaches in this paper are both index-based and ours support not only SQ but also RSQ, TIQ, and SSQ.

Besides, many skyline query variants have also been studied. We review those highly related to ours below. Spatial skyline query [24] defines spatial dominance based on object distances to individual query points (not nonspatial attributes). In [24], algorithms such as B^2S^2 and VS^2 for spatial skyline queries were extended to support an SQ by expanding a search area to cover SL. Then objects within the search area are examined. This idea is very similar to the above-discussed δ-scanning approach. Notice that



(a) Example Augmented R-tree (b) MBBs

Fig. 4. Example augmented R-tree and MBBs.

objects in SL may be spatially scattered, leading to a huge search area for SQs.

Continuous skyline query [10] treats the distances between the moving query point and all the objects as one criterion in dominance relationship and the proposed solutions [10] rely on object scan.

Dynamic skyline query [7] considers object attribute values relative to a specified object and determines nondominated objects according to that object instead of the origin. Based on dynamic skyline query, reverse skyline query [7] was defined to find all objects that have a specified object in their dynamic skylines. Both dynamic skyline query and reverse skyline query assume all the dimensions in the space are orthogonal. Differently, RSQ as introduced in this paper considers the euclidean distances that are computed according to both objects' x- and y-coordinates according to a query point at search time. Due to the different settings, algorithms designed for the reverse skyline query cannot evaluate RSQs.

3 Augmented R-Tree

Augmented R-tree (AR-tree) extends spatial R-tree [26] by maintaining for each index node the minimums of its enclosed objects' nonspatial attribute values.

Fig. 4a depicts an AR-tree with a fanout of 3. Our closely located example objects are grouped into three leaf nodes, N_1, N_2, and N_3; and their MBBs are formed as depicted in Fig. 4b. Here, N_1, N_2, and N_3 maintain the minimums of nonspatial attribute values such as (hotel price, hotel rank) of objects, namely, (100,4), (200,1) and (100,3), respectively. Unlike R-tree supporting the second BBS extension as discussed in Section 2, AR-tree can keep spatially separate objects in different MBBs. Update of an AR-tree can be performed by propagating changes from bottom up, i.e., from updated objects along tree paths toward the root [16].

On augmented R-tree, search algorithms for SQs, RSQs, TIQs, and SSQs are developed and presented in the following sections.

3.1 SQ and SSQ Evaluation Using AR-tree

An SQ (i.e., $\mathrm{SQ}(q, \mathcal{O})$) can be evaluated through an iterative NN search [9] on an AR-tree indexing $\mathcal{O}$. In this search, objects and index nodes are maintained in a priority queue P and accessed according to their shortest euclidean distances to q. If multiple queued entries are equidistant to q, one with the smallest sum of nonspatial attribute values is examined first. In details, the search in each round fetches an entry with the smallest distance to q from P and performs a dominance check on that entry against existing SQ result candidates maintained in a candidate set $\mathcal{C}$. If it is locationally dominated by any object in $\mathcal{C}$, the entry is discarded from a further examination. Otherwise, a node referred by it is expanded and all its children are enqueued into P; or an object pointed by it is inserted into $\mathcal{C}$. When P becomes vacated, the search completes and all nonlocationally dominated objects maintained in $\mathcal{C}$ are returned as the SQ result.

An SSQ (i.e., $\mathrm{SSQ}(q, \mathcal{O}, \mathcal{A}')$) can be evaluated by this approach except dominance checks examine $\mathcal{A}'$ instead.

3.2 RSQ Evaluation Using AR-Tree

If $o' \vdash_q o$, an object o' should not have a longer distance to a query point q than another object o. Based on this observation, our search algorithm for RSQs on AR-tree is developed.

Provided an RSQ (i.e., $\mathrm{RSQ}(\mathcal{Q}, o)$) issued at o, the union of all vicinity circles $cir(q, |o, q|)$ centering at individual query points $q(\in \mathcal{Q})$ and encircling o forms an initial search space. Thereafter, an AR-tree indexing $\mathcal{O}$ is traversed. During the search, those index nodes are expanded as long as their MBBs overlap any vicinity circle and their minimums of attribute values are not worse than o. Otherwise, the descendants of those index nodes are waived from detailed examinations. If any vicinity circle encloses any object that locationally dominates o, corresponding query points should not be in an RSQ result and their circles are removed from the search space. The search continues until 1) all objects that may locationally dominate o within any circle are examined, or 2) all circles are removed. Finally, the query points of those remaining circles form an RSQ result set.

Fig. 4b illustrates this search algorithm. Let us consider two query points q_1 and q_2 and an RSQ issued at o_2. Then, two vicinity circles, i.e., $cir(q_1, |o_2, q_1|)$ and $cir(q_2, |o_2, q_2|)$, form the search space. Assume that the index is traversed in a depth-first order. After the root is explored, N_1 is first expanded. Then, o_2 which is a queried object is skipped. Next, o_1 staying out of the two circles is also skipped. Thereafter, N_2 is accessed. Since $cir(q_1, |o_2, q_1|)$ encloses o_3 that locationally dominates o_2, $cir(q_1, |o_2, q_1|)$ is removed from the search space. Further, the remainders, namely, o_4 and N_3, which are outside of $cir(q_2, |o_2, q_2|)$, are all skipped. Finally, q_2 is the RSQ result.

3.3 TIQ Evaluation Using AR-Tree

To answer a TIQ, the degree of influence $I_o(\mathcal{Q})$ for each object o with respect to a set of query points $\mathcal{Q}$ needs to be estimated and then K objects with the greatest degrees of influence are outputted.

If $I_o(\mathcal{Q})$ for each object o is estimated based on $|\mathrm{RSQ}(\mathcal{Q}, o)|$, a huge number of RSQs are evaluated against $\mathcal{Q}$, making TIQs costly to answer. Rather, $I_o(\mathcal{Q})$ for each object o can be incrementally computed through evaluating SQs on individual query points q in $\mathcal{Q}$, which are expected to be smaller than $\mathcal{O}$, in practice.

This incremental search algorithm for TIQ is described as follows: First, each object o is provided with a counter $\mathtt{cnt}_o$

to keep track of the number of SQ results that include o. Initially, $\texttt{cnt}_o$ is set to 0. Afterward, SQs are evaluated, each for one query point q in $\mathcal{Q}$. Whenever o is included in an SQ result, $\texttt{cnt}_o$ is incremented by one. After all SQs are evaluated, $\texttt{cnt}_o$ indicates $I_o(\mathcal{Q})$ and K objects with the largest counter values are returned.

The efficiency of this incremental approach can be further improved by saving some SQs from being evaluated. As we can observe, some closely located query points may produce identical SQ results. That means if an SQ at a query point q has been evaluated, SQs issued at n other query points near q may share the same SQ result. In this case, the counters for those SQ result objects can be directly incremented by $n+1$. To enable a quick determination whether one SQ result for q can be shared to other SQs, we introduce the notion of *validity distance* denoted by $\mathcal{X}_{q,\mathcal{O}}$ for a given SQ result $\text{SQ}(q,\mathcal{O})$. Definition 3 formalizes this validity distance notion. Now, when any query point q' has its distance to q, i.e., $|q,q'|$, not greater than $\mathcal{X}_{q,\mathcal{O}}$, $\text{SQ}(q',\mathcal{O})$ is guaranteed to be equal to $\text{SQ}(q,\mathcal{O})$, as explained by Lemma 1. Thus, an SQ at q' can be saved from evaluation.

Definition 3 (Validity Distance). *Suppose that $o_1, o_2, \ldots o_{|\mathcal{O}|}$ in $\mathcal{O}$ are in an ascending distance order with respect to a query point q, where $i<j \Rightarrow |o_i,q| \le |o_j,q|$.*[5] *$\text{SQ}(q,\mathcal{O})$ ($\subseteq \mathcal{O}$) are SQ result objects. We denote that o has no worse attributes than o' (i.e., $\forall_{a\in\mathcal{A}} o[a] \le o'[a]$) by $o \lhd o'$. Accordingly, we formulate validity distance for $\text{SQ}(q,\mathcal{O})$ (denoted by $\mathcal{X}_{q,\mathcal{O}}$) as follows:*

$$\mathcal{X}_{q,\mathcal{O}} = \min_{o_i\in\mathcal{O}} \{\mathcal{X}_{q,\mathcal{O}}(o_i)\},$$

where $\mathcal{X}_{q,\mathcal{O}}(o_i)$ suggests a validity distance for o_i that o_i remains either nonlocationally dominated (if it is in $\text{SQ}(q,\mathcal{O})$) or locationally dominated (if it is not in $\text{SQ}(q,\mathcal{O})$). In detail, $\mathcal{X}_{q,\mathcal{O}}(o_i)$ is formulated according to three following conditions. Specifically,

$$\mathcal{X}_{q,\mathcal{O}}(o_i) = \begin{cases} \infty, & \text{if } o_i \in \text{SQ}(q,\mathcal{O}) \wedge \nexists_{i<j}\, o_j \lhd o_i, \\ \dfrac{|o_j,q|-|o_i,q|}{2}, j=\min\{j \mid i<j, o_j \lhd o_i\}, & \\ & \text{if } o_i \in \text{SQ}(q,\mathcal{O}) \wedge \exists_{i<j}\, o_j \lhd o_i, \\ \dfrac{|o_i,q|-|o_j,q|}{2}, j=\max\{j \mid j<i, o_j \in \text{SQ}(q,\mathcal{O}) \wedge o_j \lhd o_i\}, & \\ & \text{otherwise (i.e., } o_i \notin \text{SQ}(q,\mathcal{O})). \end{cases}$$

Lemma 1. *Given an SQ result $\text{SQ}(q,\mathcal{O})$, a corresponding validity distance $\mathcal{X}_{q,\mathcal{O}}$ and a query point q', that $|q,q'| \le \mathcal{X}_{q,\mathcal{O}} \Rightarrow \text{SQ}(q,\mathcal{O}) = \text{SQ}(q',\mathcal{O})$ is true.*

Proof. By triangular inequality, the distance between any object o and q', i.e., $|o,q'|$ is bounded as $|o,q|-|q,q'| \le |o,q'| \le |o,q|+|q,q'|$, according to $|o,q|$ and $|q,q'|$. Based on this, we prove this lemma by contradiction. Here, we assume that $|q,q'| \le \mathcal{X}_{q,\mathcal{O}}$ but $\text{SQ}(q',\mathcal{O}) \neq \text{SQ}(q,\mathcal{O})$ due to two following possible scenarios.

5. If multiple objects are equidistant from q, one with the smallest sum of nonspatial attributes are ordered first. Otherwise, the tie is broken arbitrarily.

The first scenario (i.e., $\text{SQ}(q,\mathcal{O}) - \text{SQ}(q',\mathcal{O}) \neq \emptyset$) occurs when a certain object o_i in $\text{SQ}(q,\mathcal{O})$ becomes locationally dominated by another object o_j (where $o_j \lhd o_i$) with respect to q'; so o_i is not included in $\text{SQ}(q',\mathcal{O})$. With respect to q, o_i is sorted before o_j. Otherwise, o_i should be excluded from $\text{SQ}(q,\mathcal{O})$. Now, because of q', $|o_j,q'| < |o_i,q'|$ that implies $|o_j,q|-|q,q'| < |o_i,q|+|q,q'|$ or $|o_j,q|-|o_i,q| < 2\cdot|q,q'|$. However, that

$$|q,q'| \le \mathcal{X}_{q,\mathcal{O}} \le \frac{|o_j,q|-|o_i,q|}{2} < |q,q'|$$

should not be true. Besides, if no such o_j exists, o_i should be included in $\text{SQ}(q',\mathcal{O})$.

The second scenario (i.e., $\text{SQ}(q',\mathcal{O}) - \text{SQ}(q,\mathcal{O}) \neq \emptyset$) implies a certain object o_i not in $\text{SQ}(q,\mathcal{O})$ becomes nonlocaitonally dominated by some other with respect to q'. Logically, there exists an object o_j (where $o_j \lhd o_i$ and $|o_j,q| < |o_i,q|$) in $\text{SQ}(q,\mathcal{O})$. As it is closer to q', o_i becomes nonlocationally dominated by o_j, i.e., $|o_i,q'| < |o_j,q'|$, which can be rewritten into $|o_i,q|-|q,q'| < |o_j,q|+|q,q'|$. Again, this results in $|q,q'| \le \mathcal{X}_{q,\mathcal{O}} \le \frac{|o_i,q|-|o_j,q|}{2} < |q,q'|$, which should not be true.

Since the two scenarios can never happen when $|q,q'| \le \mathcal{X}_{q,\mathcal{O}}$, the statement $\text{SQ}(q,\mathcal{O}) = \text{SQ}(q',\mathcal{O})$ is true. □

The computation of the validity distance $\mathcal{X}_{q,\mathcal{O}}$ for $\text{SQ}(q,\mathcal{O})$ can be incrementally computed and seamlessly integrated with SQ evaluation. The logic is outlined in Algorithm 1.

Algorithm 1. Validity distance $\mathcal{X}(q,\mathcal{O})$ computation.
Global: validity distance $\mathcal{X}_{q,\mathcal{O}}$ (initialized to ∞)
Input: an entry ϵ (an object or index node)
BEGIN
1. **if** ϵ is an index node and $\nexists_{o_i\in\mathcal{C}}\, o_i \vdash_q \epsilon$ **then return**;
2. **foreach** ($o \in \mathcal{C}$ in desc. dist. order w.r.t. q) **do**
3. **if** ($\epsilon \lhd o$) **then**
4. $\mathcal{X}_{q,\mathcal{O}} \leftarrow \min(\mathcal{X}_{q,\mathcal{O}}, \frac{|\epsilon,q|-|o,q|}{2})$;
5. **return**;

END

In the algorithm, $\mathcal{X}_{q,\mathcal{O}}$ is maintained as a global variable and initialized to ∞. Then, an entry ϵ, which can be either an object or an index node, is examined in each invocation. If an entry is an index node and it is not locationally dominated by existing result candidates in $\mathcal{C}$, the algorithm ends immediately without updating $\mathcal{X}_{q,\mathcal{O}}$ (line 1). This is because descendants enclosed by the node will be visited in subsequent invocations. Here, the result of dominance check for ϵ can be obtained from SQ evaluation. Next, SQ result candidates are compared with ϵ in a descending distance order with respect to q (lines 2-5). When ϵ has its nonspatial attributes not worse than any result object o (i.e., $\epsilon \lhd o$) in $\mathcal{C}$, $\mathcal{X}_{q,\mathcal{O}}$ is updated to $\frac{|\epsilon,q|-|o,q|}{2}$ if it is smaller than the current value of $\mathcal{X}_{q,\mathcal{O}}$ (line 4). Thereafter, the algorithm completes (line 5), as objects accessed after o cannot provide any shorter validity distance. Besides, some index nodes would be pruned during SQ evacuation, so this algorithm may provide a shorter validity distance than the actual one.

Finally, with the validity distances for individual SQs, our search algorithm (as depicted in Algorithm 2) for TIQs can run very efficiently.

Algorithm 2. TIQ Search.
Input: a set of query points $\mathcal{Q}$;
the number of req. objects K
Output: K result objects $\mathcal{O}'$ ($\subseteq \mathcal{O}$)
BEGIN
1. $\texttt{cnt}_o \leftarrow 0, \forall o \in \mathcal{O}$;
2. **while** ($\mathcal{Q} \neq \emptyset$) **do**
3. pick q from $\mathcal{Q}$; $\mathcal{Q} \leftarrow \mathcal{Q} - \{q\}$;
4. evaluate $\mathrm{SQ}(q, \mathcal{O})$ and $\mathcal{X}_{q,\mathcal{O}}$ (based on Algorithm 1);
5. $\mathcal{Q}' \leftarrow \{q' \mid q' \in \mathcal{Q}, |q, q'| \leq \mathcal{X}_{q,\mathcal{O}}\}$;
6. $\texttt{cnt}_o \leftarrow \texttt{cnt}_o + 1 + |\mathcal{Q}'|, \forall o \in \mathrm{SQ}(q, \mathcal{O})$;
7. $\mathcal{Q} \leftarrow \mathcal{Q} - \mathcal{Q}'$;
8. $\mathcal{O}' \leftarrow \{o_1, o_2, \cdots o_K \mid o_i, o_j \in \mathcal{O}, i < j \Rightarrow \texttt{cnt}_{o_i} \geq \texttt{cnt}_{o_j}\}$;
9. **output** $\mathcal{O}'$;
END

In the algorithm, the result for an SQ at each query point q and corresponding validity distance $\mathcal{X}_{q,\mathcal{O}}$ are computed (line 4). Next, all remaining query points whose distances to q are smaller than $\mathcal{X}_{q,\mathcal{O}}$ are collected into $\mathcal{Q}'$ (line 5). Then, $\texttt{cnt}_o$ for each object o is incremented by $1 + |\mathcal{Q}'|$ (line 6). The search completes after all query points are evaluated. Finally, K objects with the greatest counter values are determined and returned (lines 8-9).

4 Dominance Diagram

Dominance diagram (DD) is a solution-based approach, based on the notion of nondominance scopes. In what follows, we formulate nondominance scope and devise algorithms for 1) nondominance scope computation, 2) SQ, RSQ, SSQ, and TIQ evaluation based on indexed nondominance scopes, and 3) nondominance scope maintenance. All those algorithms can run in parallel.

4.1 Nondominance Scope Formulation

First of all, Definition 4 gives the formal definition of nondominance scopes.

Definition 4 (Nondominance Scopes). *For an object o, a nondominance scope $\mathcal{D}(o)$, represents an area ($\subseteq \mathcal{S}$) such that with respect to any point $p \in \mathcal{D}(o)$, o is not locationally dominated by any other objects, i.e., $\forall_{o' \in \mathcal{O}-\{o\}} \forall_{p \in \mathcal{D}(o)} o' \not\vdash_p o$. Based on this, $\mathcal{D}(o)$ is formulated and expressed as follows:*

$$\mathcal{D}(o) = \{p \mid p \in \mathcal{S} \wedge \nexists_{o' \in (\mathcal{O}-\{o\})}\ o' \vdash_p o\}. \tag{3}$$

On the other hand, because an infinite number of possible points in $\mathcal{S}$ and a quadratic number of object comparisons need to be examined for each possible point, computing precise nondominance scopes according to (3) is computationally infeasible. Here, we revise a nondominance scope formulation to improve the computational cost. First, we use $\mathcal{D}_{o'}(o)$ to denote a region that an object o is not locationally dominated by another object o'. Correspondingly, $\mathcal{D}(o)$ is logically treated as an intersection area among $\mathcal{D}_{o'}(o)$ for all other objects o' ($o' \neq o$) bounded by $\mathcal{S}$, as expressed in the following equation:

$$\mathcal{D}(o) = \bigcap_{o' \in \mathcal{O}-\{o\}} (\mathcal{D}_{o'}(o) \cap \mathcal{S}). \tag{4}$$

Now, let us focus on the formulation of $\mathcal{D}_{o'}(o)$. Recall (1) in Definition 1, all the three conditions should hold if o is locationally dominated by o'. In reverse thinking, o is not locationally dominated by o' if any of these conditions is violated. Accordingly, we identify three disjointed conditions that o cannot be locationally dominated by o' with respect to any point p in $\mathcal{S}$ (i.e., $o' \not\vdash_p o$) as stated in (5). Based on these conditions, $\mathcal{D}_{o'}(o)$ can be efficiently and precisely derived

$$\begin{aligned} o' \not\vdash_p o &= \neg((\forall_{a \in \mathcal{A}} o'[a] \leq o[a]) \\ &\quad \wedge (|o', p| \leq |o, p|) \\ &\quad \wedge (\exists_{a \in \mathcal{A}} o'[a] < o[a] \vee |o', q| < |o, p|)) \\ &= (\exists_{a \in \mathcal{A}} o'[a] > o[a]) \\ &\quad \vee (|o', p| > |o, p|) \\ &\quad \vee (\forall_{a \in \mathcal{A}} o'[a] \geq o[a] \wedge |o', p| \geq |o, p|) \\ &= (\exists_{a \in \mathcal{A}} o'[a] > o[a]) \\ &\quad \vee (|o', p| > |o, p|) \\ &\quad \vee (\forall_{a \in \mathcal{A}} o'[a] = o[a] \wedge |o', p| = |o, p|). \end{aligned} \tag{5}$$

These three conditions enable us to identify all possible points p, just according to the nonspatial attribute values and locations of o and o'. They are detailed as follows: Condition 1: $\exists_{a \in \mathcal{A}}\ o'[a] > o[a]$ is satisfied when o has some nonspatial attributes better than o', no matter o' is closer to p than o or not. With this condition, an entire space $\mathcal{S}$ is implied to contribute $\mathcal{D}_{o'}(o)$. Condition 2: $|o', p| > |o, p|$ is true, if p is closer to o than o'. This condition implies an area covered by a half-plane $H_{o,o'}$ [20] but not the perpendicular bisector $B_{o,o'}$ [20] constitutes $\mathcal{D}_{o'}(o)$. Here, $H_{o,o'} - B_{o,o'}$ covers all locations closer to o than o' and $B_{o,o'}$ refers to those locations equidistant to o and o'. Condition 3: $\forall_{a \in \mathcal{A}}\ o'[a] \geq o[a] \wedge |o', p| \geq |o, p|$ can be further rewritten into $\forall_{a \in \mathcal{A}}\ o'[a] = o[a] \wedge |o', p| = |o, p|$, because of partial overlaps with the former two conditions. This condition says both o and o' have equal values for all nonspatial attributes and they are equally close to some locations, implying p on the perpendicular bisector $B_{o,o'}$ between o and o'.

Based on the above analysis, $\mathcal{D}_{o'}(o)$ for an object o with respect to another object o' consists of three components, namely, $\mathcal{D}_{1o'}(o)$, $\mathcal{D}_{2o'}(o)$, and $\mathcal{D}_{3o'}(o)$; and it is formally stated as follows:

$$\mathcal{D}_{o'}(o) = \mathcal{D}_{1o'}(o) \cup \mathcal{D}_{2o'}(o) \cup \mathcal{D}_{3o'}(o), \tag{6}$$

where

$$\mathcal{D}_{1o'}(o) = \begin{cases} \mathcal{S}, & \text{if } \exists_{a \in \mathcal{A}}\ o'[a] > o[a], \\ \emptyset, & \text{otherwise.} \end{cases}$$

$$\mathcal{D}_{2o'}(o) = (H_{o,o'} - B_{o,o'}), \text{and}$$

$$\mathcal{D}_{3o'}(o) = \begin{cases} B_{o,o'}, & \forall_{a \in \mathcal{A}}\ o[a] = o'[a], \\ \emptyset, & \text{otherwise.} \end{cases}$$

Let us exemplify how nondominated scopes are formed for our example objects. As it has the lowest price, o_1 is not dominated by other objects except o_5 that has equal values in nonspatial attributes, $\mathcal{D}(o_1)$ includes both H_{o_1,o_5}

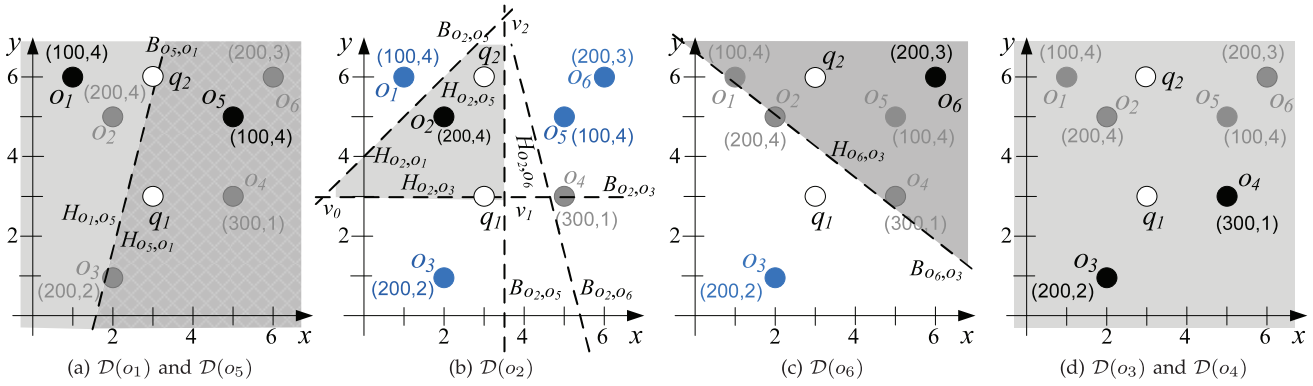


Fig. 5. Formation of nondominance scopes.

and B_{o_5,o_1}, according to Conditions 2 and 3, respectively. Symmetrically, $H_{o_5,o_1} \cup B_{o_5,o_1}$ forms $\mathcal{D}(o_5)$. Both $\mathcal{D}(o_1)$ and $\mathcal{D}(o_5)$ are depicted in Fig. 5a.

Next, o_2 has no nonspatial attributes better than o_1, o_3, o_5, and o_6, so $\mathcal{D}(o_2)$ as shown in Fig. 5b is formed as the intersection area of $H_{o_2,o_i} - B_{o_2,o_i}$, where $o_i \in \{o_1, o_3, o_5, o_6\}$. Further, o_6 is only dominated by o_3. Hence, $\mathcal{D}(o_6)$ equals $H_{o_6,o_3} - B_{o_6,o_3}$, as shown in Fig. 5c. Finally, o_3 and o_4, which are not dominated by any other objects, have an entire space as their nondominance scopes. Fig. 5d depicts $\mathcal{D}(o_3)$ and $\mathcal{D}(o_4)$.

An object o can obtain an empty nondominance scope, i.e., $\mathcal{D}(o) = \emptyset$ if 1) another object o' is colocated at o and 2) o' has all nonspatial attributes not worse than o and one or more attributes strictly better than o. In this case, $\mathcal{D}(o)$ needs not computed and maintained.

To represent $\mathcal{D}(o)$, which is an n-vertex convex polygon, we maintain a list of n vertexes: $v_0, \ldots v_{n-1}$, in a counterclockwise order around o. The ith edge connects two adjacent vertexes, i.e., $(v_i, v_{(i+1) \bmod n})$ for $i \in [0, n-1]$. Refer to Fig. 5, $\mathcal{D}(o_2)$ is composed of vertexes v_0, v_1, and v_2. Besides, an n-bit vector is stored, in which the ith bit indicates whether the ith edge (i.e., a segment of a perpendicular bisector) is included as a part of $\mathcal{D}(o)$. In the figure, edges bounding $\mathcal{D}(o_2)$ are not included, and thus all the corresponding bits are set to off. To determine whether a given point p is inside $\mathcal{D}(o)$ (i.e., $p \in \mathcal{D}(o)$), we can check if p is on the left side of every edge $(v_i, v_{(i+1) \bmod n})$ if the ith bit is off, or if p is not on the right side of every $(v_i, v_{(i+1) \bmod n})$ if the ith bit is on.

4.2 Nondominance Scope Computation

Despite (4) provides an equivalent and more efficient nondominance scope formulation than (3), it would still incur a lot of object comparisons. In this section, we derive and present an efficient nondominance scope computation algorithm equipped with some optimization strategies, according to (4).

To efficiently compute nondominance scopes, we devise four following strategies:

1. *Parallelizing nondominance scope computation.* Nondominance scopes for objects can be independently derived. Thus, the computation can be parallelized in available processors to shorten the elapsed time.
2. *Examining objects with no worse attributes only.* According to the first condition of (6), any object o' with any nonspatial attributes worse than o contributes an entire space $\mathcal{S}$ to $\mathcal{D}_{o'}(o)$. Thus, comparisons between o and o' can be saved. Our second optimization strategy is that only those with all attributes not worse than o are accessed.
3. *Exploiting largest empty circle property.* At each vertex v of a nondominance scope $\mathcal{D}(o)$ for an object o, a circumcircle $cir(v, |v, o|)$ whose radius is $|v, o|$ is formed according to two or more objects that effectively affect the formation of $\mathcal{D}(o)$. If they are the actual vertexes of $\mathcal{D}(o)$, these circumcircles should only include any other objects whose attributes are all better than or equal to o. Fig. 6 illustrates $\mathcal{D}(o_2)$'s three circumcircles centering at vertexes v_0, v_1, and v_2. The radii of these circumcircles are $|v_0, o_2|$, $|v_1, o_2|$, and $|v_2, o_2|$. $\mathcal{D}(o_2)$ is formed since $cir(v_0, |v_0, o_2|)$, $cir(v_1, |v_1, o_2|)$, $cir(v_1, |v_1, o_2|)$ are empty. This is the so-called *largest empty circle property*.

   Accordingly, our third strategy is to exploit this largest empty circle property to eliminate detailed comparisons for certain objects. In our example, after o_5 is examined, $\mathcal{D}(o_2)$ is formed as shown in Fig. 6. Since o_6 is just on the border but not enclosed by any circumcircle, o_6 can be completely ignored in refining $\mathcal{D}(o_2)$.
4. *Comparing objects in distance order.* $\mathcal{D}(o)$ for an object o is mostly determined by those objects near o. Therefore, our fourth strategy suggests to examine objects according to their nearness to o so as to refine o's nondominance scope. Additionally, we check $|o', o|$ against the diameters of all largest empty circles (i.e, $2 \times |v, o|$ for all vertexes v) of currently

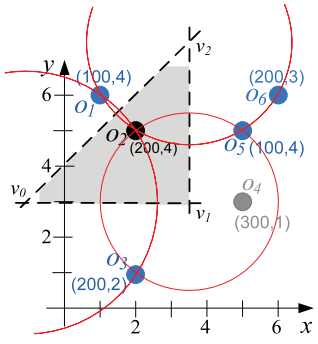


Fig. 6. Largest empty circles.

formed $\mathcal{D}(o)$. If $|o', o| > \max_v\{2 \times |v, o|\}$, o' and subsequent objects are guaranteed to be outside of all circumcircles and o can be skipped from examination. Thereafter, computation $\mathcal{D}(o)$ is done and no further refinement is needed.

Based on all the discussed optimization strategies, a nondominance scope computation algorithm is developed and its pseudocode is outlined in Algorithm 3.

Algorithm 3. Nondominance Scope Computation.
Input: an object set $\mathcal{O}$; an entire space $\mathcal{S}$;
the root of an aggregated R-tree r;
the number of processors num;
Output: a set of nondominance scopes D;
BEGIN
1. divide $\mathcal{O}$ evenly into $\mathcal{O}_1 \cup \mathcal{O}_2 \cup \cdots \mathcal{O}_{num}$;
2. **parallel foreach** $i \in [1, num]$ **do** *// opt. 1*
3. **foreach** $o \in \mathcal{O}_i$ **do**
4. $\mathcal{D}(o) \leftarrow \mathcal{S}$; P.enqueue(r);
5. **while** (P is not empty) **do** *// opt. 4*
6. $\epsilon \leftarrow P$.dequeue();
7. **if** $|o, \epsilon| > \max_{v \in \text{vertex}(\mathcal{D}(o))} 2 \times |v, o|$ **then goto** 15; *// opt. 4*
8. **if** (ϵ is o) **return** 5;
9. **if** $\exists_{a \in \mathcal{A}} o[a] < \epsilon[a]$ **then goto** 5; *// opt. 2*
10. **if** $\forall_{v \in \text{vertex}(\mathcal{D}(o))} |\epsilon, v| \geq |o, v|$ **then** goto 5; *// opt. 3*
11. **if** ϵ is an object **then**
12. $\mathcal{D}(o) \leftarrow \mathcal{D}(o) \cap (\mathcal{D}_{1_\epsilon}(o) \cup \mathcal{D}_{2_\epsilon}(o) \cup \mathcal{D}_{3_\epsilon}(o))$;
13. **else** *// ϵ is an index node*
14. P.enqueue(ϵ_c), $\forall \epsilon_c \in child(\epsilon)$;
15. $D \leftarrow D \cup \{\mathcal{D}(o)\}$;
16. **return** D;
END

The algorithm first divides $\mathcal{O}$ evenly into num object sets, $\mathcal{O}_i$, where $i \in [1, num]$ (line 1) according to num, the number of available processors. Then, it computes nondominance scopes for objects in different subsets in parallel (lines 2-15). Next, the nondominance scope $\mathcal{D}(o)$ for each object o in $\mathcal{O}_i$ is computed (lines 3-14). Initially, $\mathcal{D}(o)$ is set to $\mathcal{S}$ and a priority queue P is used to order the accesses of index nodes and objects as entries in nondescending distance order. Further, an entry ϵ is fetched from P and examined according to different conditions. Whenever ϵ is outside all its circumcircles, computation of $\mathcal{D}(o)$ completes (line 7). If ϵ is o (line 8), ϵ has no better attributes (line 9) or ϵ is not enclosed by any circumcircle (line 10), a detailed examination of ϵ is skipped. Further, ϵ is used to refine current $\mathcal{D}(o)$ if ϵ is an object. Otherwise (ϵ is an index node), all its children are enqueued into P for further investigation. Finally, all derived nondominance scopes are collected in D and D is outputted at the end of the computation.

4.3 SQ and RSQ Evaluation on DD

With nondominance scopes, evaluations of SQs and RSQs can be simplified as comparing individual nondominance scopes and query points. Notice that this simplified execution effectively improves evaluation performance. Besides, query evaluation can be performed in parallel on multiple processors.

To evaluate an SQ issued at a query point q, nondominance scopes are checked against a query point q. The objects of those nondominance scopes that cover q form the SQ result set. To facilitate SQ evaluation, nondominance scopes for all objects can be indexed by a spatial index, for example, R-tree. Then, the index is traversed to find those nondominance scopes covering q. Further, nondominance scopes can be maintained in multiple indexes, based on which searches can be performed in parallel to answer SQs.

Likewise, an RSQ can be answered by checking which query points are covered by the nondominance scope for a specified object. Some indexes on query points can be built in advance to speed up the search. Again, query points can be divided into different groups to enable parallel RSQ evaluation.

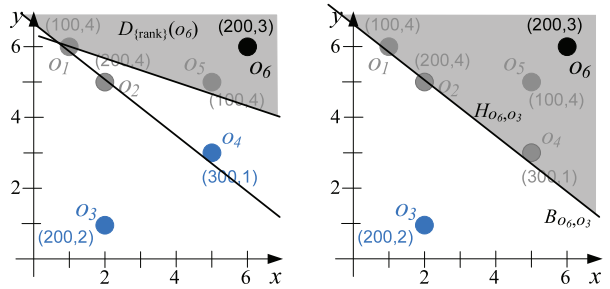
4.4 TIQ Evaluation on DD

Although the above algorithms for SQs (RSQs) can be trivially utilized to deduce the degree of influence for individual objects, it can possibly incur a huge number of SQ (RSQ) evaluations and a lot of comparisons between query points and nondominance scopes.

In this section, we develop a more efficient algorithm that can effectively avoid many comparisons and quickly narrow down the search space for the most influential objects. First of all, we assume that 1) all query points are indexed by a sum R-tree ($T_\mathcal{Q}$), which counts for each index node the number of query points indexed by the node, and 2) all nondominance scopes are indexed by an R-tree ($T_\mathcal{O}$).

Algorithm 4. TIQ Search on Dominance Diagram.
Input: the root of R-tree $T_\mathcal{O}$ on $\mathcal{D}(o)$'s, r_D;
the root of sum R-tree $T_\mathcal{Q}$ on query points $r_\mathcal{Q}$;
the requested number of result objects K;
Local: priority queue P (initialized to $\emptyset$);
Output: K result objects $\mathcal{O}'$;
BEGIN
1. enqueue $(r_D, \{r_\mathcal{Q}\})$ to P;
2. **while** P is not empty and $|\mathcal{O}'| < K$ **do**
3. $(\epsilon_D, Q) \leftarrow P$.dequeue();
4. $Q' \leftarrow \{\epsilon_Q \mid \epsilon_Q \in Q, \epsilon_Q - \epsilon_D \neq \emptyset\}$;
5. **if** $|Q'| > 0$ **then**
6. P. enqueue $(\epsilon_D, \{\epsilon'_Q \mid \epsilon'_Q \in child(\epsilon_Q), \epsilon_Q \in Q' \wedge \epsilon_Q \cap \epsilon_D \neq \emptyset\} \cup (Q - Q'))$;
7. **else if** (ϵ_D is an index node) **then**
8. P.enqueue (ϵ'_D, Q), $\forall \epsilon'_D \in child(\epsilon_D)$;
9. **else**
10. $\mathcal{O}' \leftarrow \mathcal{O}' \cup \{o\}$, where o corresponds to ϵ_D;
11. **output** $\mathcal{O}'$;
END

The search as outlined in Algorithm 4 performs the best first search. Here, a priority queue P maintains entries in form of (ϵ_D, Q), where ϵ_D can be an index node of $T_\mathcal{O}$ or a nondominance scope and Q is a set of index nodes of $T_\mathcal{Q}$ on query points and/or individual query points covered by ϵ_D. The ordering of entries in P is determined by their numbers of involved query points. In doing so, nondominance scopes that are likely to cover the most query points are accessed first. Initially P is enqueued with an entry containing the roots of both $T_\mathcal{O}$ and $T_\mathcal{Q}$, i.e., $(r_D, \{r_\mathcal{Q}\})$. Then, in each iteration (lines 2-10), an entry (ϵ_D, Q) is fetched from P and



(a) $\mathcal{D}_{\{\text{rank}\}}(o_6)$ (b) $\mathcal{E}(o_6) = H_{o_6,o_3}$

Fig. 7. $\mathcal{D}_{\{\text{rank}\}}(o_6)$ and $\mathcal{E}(o_6)$.

examined. To provide a precise estimated degree of influence for ϵ_D, any index nodes of T_Q partially covered by ϵ_D, represented by $\mathcal{Q}'$ are expanded and their children are queued for later investigation (line 6). If no such $\mathcal{Q}'$ exists, ϵ_D is expanded if ϵ_D is an index node (line 8). Otherwise (i.e., ϵ_D is a nondominance scope) its degree of influence equals to the number of query points covered in $\mathcal{Q}$. Then, the corresponding object o of ϵ_D is inserted to a result set $\mathcal{O}'$ (line 10). When no more entries need to be examined or K result objects are obtained (line 2), the search terminates and $\mathcal{O}'$, i.e., result objects, are returned (line 11).

4.5 SSQ Evaluation on DD

It is quite intuitive to determine a nondominance scope $\mathcal{D}_{\mathcal{A}'}(o)$ for every object o with respect to every possible subset of attributes $\mathcal{A}'$ ($\mathcal{A}' \subset \mathcal{A}$) as to support SSQ($q, \mathcal{O}, \mathcal{A}'$). However, the number of nondominance scopes exponentially grows with the number of nonspatial attributes.

Here, we exploit the notion of *nondominance scope envelops* to determine SSQ result candidates and perform comparisons among candidate objects to compute an SSQ result. Notice that $\mathcal{D}_{\mathcal{A}'}(o) \subseteq \mathcal{D}(o)$ does not hold for all cases. For instance, $\mathcal{D}_{\{\text{rank}\}}(o_6)$, which is a nondominance scope formed for hotel o_6 with respect to the nonspatial attribute "rank," as shown in Fig. 7a, is not entirely covered by $\mathcal{D}(o_6)$. Thus, query points on the border of $\mathcal{D}_{\{\text{rank}\}}(o_6)$ are not actually covered by $\mathcal{D}(o_6)$.

As formally defined in Definition 5, a nondominance scope envelop $\mathcal{E}(o)$ covers nondominance scopes $\mathcal{D}_{\mathcal{A}'}(o)$ for an object o with respect to all possible $\mathcal{A}'$ ($\subseteq \mathcal{A}$). As explained in Lemmas 2 and 3, that $\mathcal{D}_{\mathcal{A}'}(o)$ is fully covered by its envelop $\mathcal{E}(o)$ always holds.

Definition 5 (Nondominance Scope Envelop). *For an object o ($\in \mathcal{O}$) and $\mathcal{A}$, a nondominance scope envelop, $\mathcal{E}(o)$, is an area covering o's nondominance scope $\mathcal{D}(o)$, as formulated in the following equation:*

$$\mathcal{E}(o) = \bigcap_{o' \in \mathcal{O} - \{o\}} (\mathcal{E}_{o'}(o) \cap \mathcal{S}). \quad (7)$$

Here, $\mathcal{E}_{o'}(o) = \mathcal{D}_{o'}(o) \cup H_{o,o'}$.

Lemma 2. *For any object o, the statement that $\mathcal{D}(o)$ should be contained by $\mathcal{E}(o)$ holds.*

Proof. According to Definition 5 and (6), $\mathcal{D}_{o'}(o) \subseteq \mathcal{E}_{o'}(o)$, $\forall o' \in \mathcal{O} - \{o\}$; and hence $\mathcal{D}(o) \subseteq \mathcal{E}(o)$. □

Lemma 3. *Given an object o and a subset of attributes $\mathcal{A}'$, a nondominated scope envelop $\mathcal{E}_{\mathcal{A}'}(o)$ formed with respect to $\mathcal{A}'$ should be contained by $\mathcal{E}(o)$.*

Proof. Based on the attributes values of o and another object o', there are two cases. Case 1: $\exists a \in \mathcal{A}'$ such that $o[a] < o'[a]$. Then, $\mathcal{D}_{\mathcal{A}',o'}(o) = \mathcal{D}_{o'}(o) = \mathcal{S} \Rightarrow \mathcal{E}_{\mathcal{A}',o'}(o) = \mathcal{E}_{o'}(o)$ (where $\mathcal{E}_{\mathcal{A}',o'}(o)$ is a partial $\mathcal{E}_{\mathcal{A}'}(o)$ formed with respect to o'). Case 2: $\forall a \in \mathcal{A}', o[a] \geq o'[a], \mathcal{D}_{o'}(o) = \emptyset \Rightarrow \mathcal{E}_{\mathcal{A}',o'}(o) = H_{o,o'}$. Given the fact that $\mathcal{E}_{o'}(o) = \mathcal{D}_{o'}(o) \cup H_{o,o'}$, $\mathcal{E}_{\mathcal{A}',o'}(o) \subseteq \mathcal{E}_{o'}(o)$. Both cases shows that $\mathcal{E}_{\mathcal{A}',o'}(o) \subseteq \mathcal{E}_{o'}(o)$ and hence $\mathcal{E}_{\mathcal{A}'}(o) \subseteq \mathcal{E}(o)$. □

Algorithm 5. SSQ-Search on Dominance Diagram.

Input: the root of an R-tree (r); a query point (q); a subset of attributes $\mathcal{A}'$
Local: a stack (ST);
Output: a set of result objects ($\mathcal{C}$);
BEGIN
1. ST.push(r); $\mathcal{C} \leftarrow \emptyset$;
2. **while** ($ST \neq \emptyset$) do
3. $\epsilon \leftarrow ST$.pop();
4. **if** ($q \notin \epsilon$'s MBB) **then go to** 2;
5. **if** (ϵ is a node N) **then** ST.push(c), $\forall c \in child(N)$;
6. **elseif** ($q \in \mathcal{E}(o)$, o is referred by ϵ) **then** $\mathcal{C} \leftarrow \mathcal{C} \cup \{o\}$;
7. $\mathcal{C} \leftarrow \{o_i | o_i \in \mathcal{C}, \nexists_{o_j \in \mathcal{C} - \{o_i\}} o_j \vdash_q o_i\}$;
8. **output** C;

END

With nondominance scope envelops, a search algorithm for an SSQ issued at q is developed and its pseudocode is outlined in Algorithm 5. In the algorithm, an R-tree that indexes nondominance scope envelops is traversed in depth-first order and objects whose envelops cover q are collected into a candidate set C (lines 1-6). Next, result candidates are ordered based on certain monotonic scores (e.g., the euclidean distances to q). Then, those not subspace locationally dominated are collected (line 7) and outputted (line 8). The algorithm can be modified to run in parallel in two rounds. First, each processor collects a set of result candidates. Then, each processor examines each result candidate and discards it if it is subspace locationally dominated.

4.6 Nondominance Scope Maintenance

This section studies and addresses the issue of nondominance scope maintenance. Without loss of generality, we consider two major types of updates as 1) object deletion, and 2) object insertion. Object modification can be treated as deletion of an object followed by insertion of another object with a modified position and/or nonspatial attribute values. Here, we adopt an invalidate-and-recompute strategy that nondominance scopes affected by an update are identified and then recomputed according to an updated object set.

To quickly identify affected nondominance scopes due to object deletion, each object o is maintained with references to other objects whose nondominance scopes are formed based on o. Specifically, the nondominance scopes for those objects have o encircled in circumcircles centering at certain

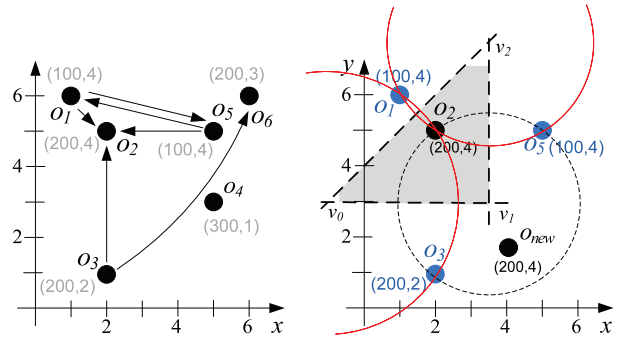


Fig. 8. References and object insertion.

vertices. Those references can be determined during the computation of nondominance scopes. When an object o_{old} is deleted, all o_{old}'s referenced objects need to recompute their nondominance scopes. Fig. 8a depicts the references of our example objects. Thus, if o_5 is deleted, it can be quickly determined that $\mathcal{D}(o_1)$ and $\mathcal{D}(o_2)$ are affected. Since one reference corresponds to one edge of an object's nondominance scope, the storage cost for references is linear to that for nondominance scopes.

When a new object o_{new} is added, affected nondominance scopes need refined. Notice that the objects of those affected nondominance scopes should not have better nonspatial attributes than o_{new}, and some circumcircles of their nondominance scopes should enclose o_{new}. Therefore, only those objects with attributes worse than or equal to o_{new} are examined and their examinations are ordered according to their shortest distances of their vertices to o_{new}. When o_{new} is covered by any circumcircle of a nondominance scope $\mathcal{D}(o')$, $\mathcal{D}(o')$ is invalidated and refined with respect to o_{new}. As depicted in Fig. 8b, a new object o_{new} has the same nonspatial attributes as o_2 and it is covered by a circumcircle of $\mathcal{D}(o_2)$. Thus, o_2 recomputes its nondominance scope. In the mean time, other objects that previously determined $\mathcal{D}(o')$ need to update their references according to this update.

Since nondominance scope envelops are derived in the same way as nondominance scopes, the discussed maintenance mechanisms can be directly utilized for nondominance scopes. Finally, after nondominance scopes are updated, updated MBBs are propagated in an R-tree to finish the update.

5 Performance Evaluation

This section reports the performance evaluation of our proposed approaches, namely, augmented R-tree (ARTree) and dominance diagram (DD). In the evaluation, *elapsed time*, the length of the time duration from when a query is issued to the time an entire result set is returned, is measured as the performance metric.

Our experiments were conducted primarily on synthetic data sets to test the scalability and sensitivity of our approaches to three major factors, namely, 1) *data set cardinality* (i.e., $|\mathcal{O}|$), 2) *number of nonspatial attributes (dimensionality)* ($|\mathcal{A}|$), and 3) *nonspatial attribute value distribution*. The synthetic data sets consisted of 10k, 50k, 100k, 500k, and 1,000k (i.e., one million) uniformly distributed objects. We also included four real data sets, namely, 1) mall, 2) school, 3) church, and 4) whrrl to test the practicality of our approaches. The data sets mall, school, and church (extracted from TIGER/Line) contained the locations of 11,989 shopping malls, 46,676 schools and 108,928 churches, respectively, in the continents of the US.[6] And, whrrl consisted of 171,620 points of interest crawled from www.whrrl.com [28], a location-based social network website. The geographical spaces for all data sets were normalized to a square area of [10,000, 10,000]. Each object was added with 2, 4, 6, or 8 randomly generated nonspatial attributes. In anticorrelated (anti), independent (indp), and correlated (corr) distributions [4], attribute values were generated independently of object positions. In spatially correlated (sp.corr) distribution, smaller nonspatial attribute values were assigned to those objects closer to the center of the space and were in correlated distribution. Each value was bounded in an integer range between 1 and V, where V was varied among 10, 50, 100, 500, and 1,000. Table 1 lists the experiment parameters.

TABLE 1
Experiment Parameters

Parameter	Values
Cardinality $\|\mathcal{O}\|$	mall: 11,989; school: 46,675 church: 108,928; whrrl: 171,620 synthetic: 10k, 50k, 100k, 500k 1,000k
Dimensionality $\|\mathcal{A}\|$	2, 4, 6, and 8
Distribution of $\mathcal{A}$	anti-correlated (anti), independent (indp), correlated (corr), and spatially correlated (sp.coor),
Max Att. Value V	10, 50, 100, 500, 1000

Those underlined values are default values.

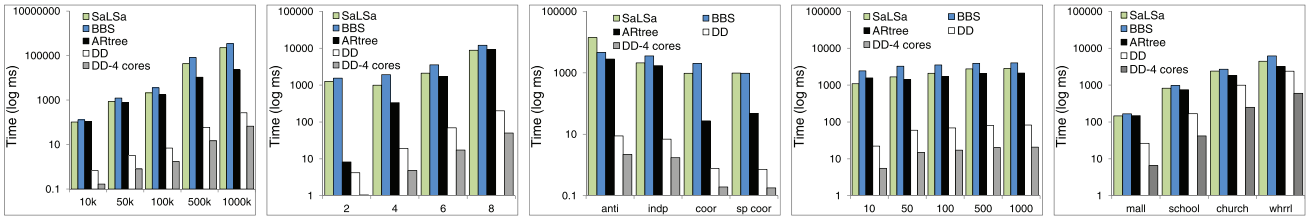
We implemented ARTree and DD as well as two representative state-of-the-art approaches, namely, SaLSa and BBS (implemented as the second extension, discussed in Section 2) for comparison, all in GNU C++. We used TGS bulk loading algorithm [8] to generate disk-based R-trees for BBS, ARTree, and DD. The maximum size of an index node (i.e., disk page) was fixed at 4 KB. To perform parallel query evaluation based on DD and nondominance scope computation and maintenance, we used POSIX threads [5] (for a single computer) and open MPI (for a computer cluster).[7] All experiments were run on 64-bit Linux computers with Intel Core i5 2.4-GHz quadcore CPUs, 64-GB RAM, and SCSI hard drives.

The following subsections report the settings and the performance of our proposed approaches in answering SQs, RSQs, TIQs, and SSQs, and the overhead incurred for the nondominance scope computation and update.

5.1 Performance Evaluation on SQ

The first experiment set evaluates the performance of ARTree, DD, SaLSa, and BBS for SQs. SaLSa examines objects sorted in nondescending order of their minimum of

6. Tiger/Line: www.census.gov/geo/www/tiger.
7. Open MPI: http://www.open-mpi.org.



(a) Cardinality $|\mathcal{O}|$ (b) Dimensionality $|\mathcal{A}|$ (c) Distribution of $\mathcal{A}$ (d) Domain Range 1-V (e) Real datasets

Fig. 9. SQ performance.

nonspatial attributes and distances to the query point. For BBS, objects are indexed on their nonspatial attributes and their aggregated spatial coordinates are maintained in index nodes in R-trees. We evaluate the parallel version of DD (DD-4 cores) as a process running four threads on a quadcore CPU.[8] In each setting, we run 100 SQs at different query points. Fig. 9 presents experiment results in terms of average elapsed time.

Figs. 9a, 9b, 9c, and 9d show elapsed time (in milliseconds) with respect to the four parameters, namely,

1. object set cardinality $|\mathcal{O}|$,
2. dimensionality $|\mathcal{A}|$,
3. distribution of nonspatial attributes, and
4. value domain range, respectively.

Unless being varied, parameters, $|\mathcal{O}|$, $|\mathcal{A}|$, distribution of $\mathcal{A}$ and V were defaulted at 100k, 6, indp, and 100, respectively.

As we can anticipate, all the approaches took longer times when data sets with larger cardinality, higher dimensionality, anticorrelated (even independent) attribute distribution were experimented. However, no significant differences are observed when V was varied. This is because in high dimensional space (six attributes in independent distribution), objects are sparsely located in the space and so many of them are not likely to be locationally dominated, regardless of nonspatial attribute value ranges. Since it is not significant to the performance, this factor is ignored from the rest of the experiments.

Specifically, BBS runs slower than SaLSa, because 1) R-tree maintains spatially far apart objects sharing similar nonspatial attribute values in MBBs, which results in a large number of false hits in BBS, 2) R-tree performance deteriorates when data set dimensionality is high, and 3) extra index access overhead relative to SaLSa. ARtree can perform slightly faster than SaLSa especially when larger data set and lower dimensionality were evaluated, since it could discard nonresult candidates from detailed examination. DD and DD-4 cores, on the other hand, perform at least an order of magnitude faster than BBS, SaLSa and ARTree that provide similar performance. It is because no expensive dominance checks during searches. Further, DD-4 cores exhibits an improvement over DD by parallelizing searches.

Fig. 9e reports the elapsed time for all the approaches on real data sets. Due to a large number of attributes (defaulted at 6) and indp value distribution, many objects were not locationally dominated. Thus, high computational cost (for dominance checks) and I/O cost were resulted in SaLSa, BBS, and ARTree. Relative to SaLSa (the only nonindexed approach in the experiments), the average improvement of BBS, ARTree, DD and DD-4 cores is 28, 120, 370, and 1,480 percent, respectively. These numbers indicate the outperformance of DD and DD-4 cores in support of SQs.

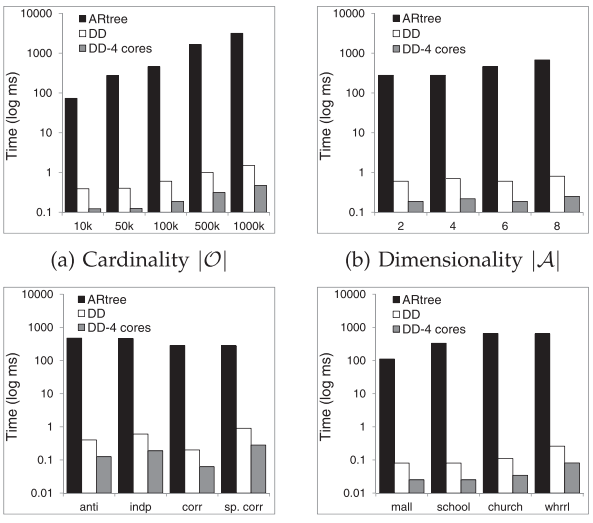
8. In the implementation, a pool of four worker threads are first created (through POSIX threading); and all the threads evaluate an SQ based on subsets of objects in separate indexes. An SQ result is collected as the union of results from all the threads.

5.2 Performance Evaluation on RSQ

The second experiment set evaluates the performance of ARTree, DD, and DD-4 cores in supporting RSQs. In the experiments, we selected 100 objects, at which we evaluated RSQs against 1,000 randomly selected query points. The experiment results are shown in Fig. 10.

Figs. 10a, 10b, and 10c show results about the impacts of $|\mathcal{O}|$, $|\mathcal{A}|$, and the attribute value distribution, respectively. Again, DD and DD-4 cores show superior performance, more than two orders of magnitude faster than ARTree. DD-4 cores is the most efficient approach.

In detail, ARTree, DD, and DD-4 cores took longer elapsed time for larger data sets, due to higher index access costs (see Fig. 10a). When $|\mathcal{A}|$ increases, a slightly longer elapsed time was resulted, because of higher computational cost incurred for each dominance check (see Fig. 10b). Conversely, the performance of ARTree, DD, and DD-4 cores does not vary much for different value distributions, since ARTree incurs the same amount of dominance checks with respect to the same data set size; and DD and DD-4 cores check the same number of query



(a) Cardinality $|\mathcal{O}|$ (b) Dimensionality $|\mathcal{A}|$

(c) Distribution of $\mathcal{A}$ (d) Real datasets

Fig. 10. RSQ performance.

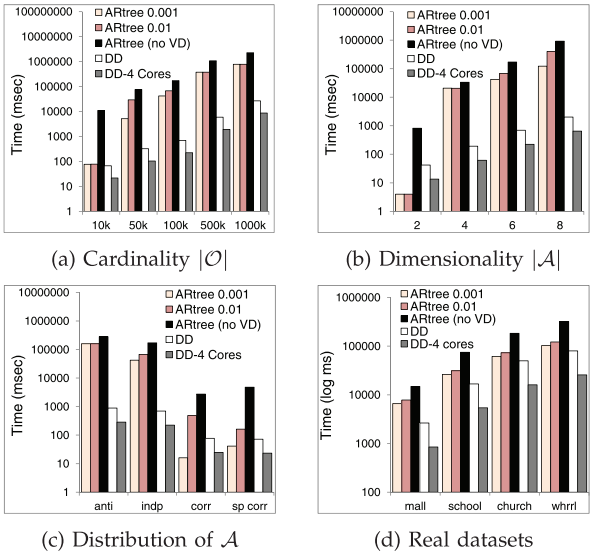


Fig. 11. TIQ performance ($K = 100$).

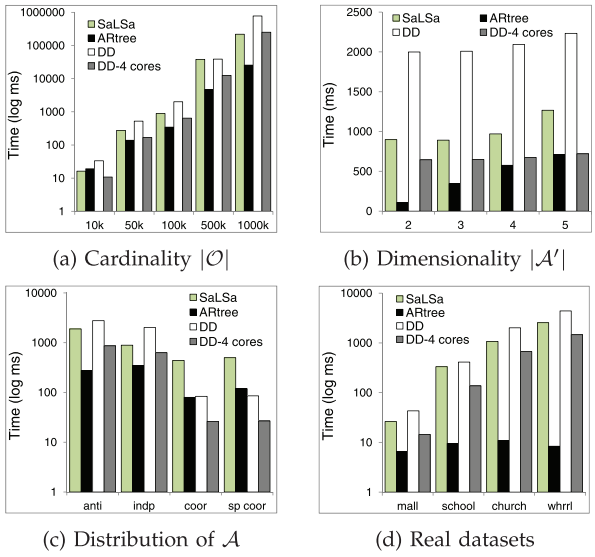


Fig. 12. SSQ performance.

points against a nondominance scope (see Fig. 10c). Overall, DD performs much faster than ARTree, because DD needs no dominance checks.

The experiment results on real data sets are depicted in Fig. 10d. DD, which took less than 0.1 ms in most of the data sets, exhibits at least three orders of magnitude faster than ARTree, which consumed, on average, 432.5 ms. Further, DD-4 cores spent about 0.04 ms on processing an RSQ in parallel.

5.3 Performance Evaluation on TIQ

The third set of experiments evaluate the performance of ARTree and DD in supporting TIQs with K set to 100. We also include ARTree (no VD) that does not use validity distances to save some SQ evaluations, and DD-4 cores that first determines local top K result candidates from four subsets of objects and derives the global top K result from those local results. Besides, we generated two sets of 1,000 uniformly distributed query points, where one query set is bounded by a square subspace of [0.001,0.001] and the other by [0.01,0.01]. The query points in subspace of [0.001,0.001] are spatially closer to each other than those in subspace in [0.01,0.01].

Since ARTree (no VD) evaluates all SQs and DD and DD-4 cores access most objects before K result objects can be determined, their performance is the same for the two query point sets; and thus we report their results on query point sets bounded by [0.001,0.001] only.

We varied the data set cardinality, dimensionality and distribution of attributes and the results are shown in Figs. 11a, 11b, and 11c, respectively. ARTree 0.001 and ARTree 0.01 that run ARTree with validity distance on query point sets in [0.001,0.001] and [0.01,0.01], respectively, run considerably faster than ARTree (no VD), especially when smaller data sets, lower dimensionalities and correlated value distributions were experimented. DD and DD-4 cores, on the other hand, run faster than others, because of their fast SQ evaluation. Fig. 11d plots the results on real data sets. On average, ARTree 0.001, ARTree 0.01, ARTree (no VD), DD, and DD-4 cores spent 49.3, 58.5, 148.7, 37.2, and 12.0 seconds.

5.4 Performance Evaluation on SSQ

The fourth set of experiments study the performance of ARTree and DD in SSQ evaluations. Again, we included DD-4 cores, which filters SSQ result candidates followed by comparing candidates to determine a final SSQ result set. We also included SaLSa, which has been shown to perform faster than BBS in SQs, for comparison. Here, the data set dimensionality was fixed at 6. By default, three nonspatial attributes were queried (i.e., $|\mathcal{A}'| = 3$) while other parameters were varied.

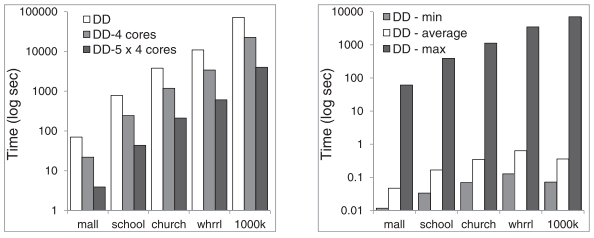
Very different from the previous experiments, ARTree performs faster than both DD and DD-4 cores as well as SaLSa, for synthetic data sets with different cardinality (see Fig. 12a) and attribute distributions (see Fig. 12c). For coor and sp.corr, DD-4 cores performs slightly better than ARTree, because fewer candidates are examined. Meanwhile, the performance of ARTree considerably deteriorates when higher dimensionality was considered (see Fig. 12b).

As shown in Fig. 12d, ARTree spends 9 ms while DD and DD-4 cores take on average 1,722 and 538 ms, respectively, on real data sets. This can be explained that DD and DD-4 cores retrieve many result candidates, according to nondominance scope envelops, and perform costly dominance check among those candidates.

5.5 Indexing Overhead

The final experiment set investigates the overhead of computing and updating nondominance scopes. Fig. 13a shows elapsed time taken to compute nondominance scopes for real data sets and the largest synthetic data sets with 1,000k objects. In addition to DD and DD-4 cores, we implemented DD-5×4 cores that computes nondominance scopes on a small cluster of five computers with quadcore CPUs and the implementation was based on open MPI. As shown, DD-5×4 cores took 3.8 seconds for the smallest data set mall, about 3.5 and 10 minutes for larger data sets church and whrrl, respectively, and about 66.3 minutes for the largest synthetic data set.

Next, we examine nondominance scope update costs. In the experiments, we selected 1,000 objects to delete and



(a) Computation time (b) Maintenance time

Fig. 13. Index computation and maintenance cost.

reinserted them in random positions near their original locations. In the object selection, we divided all objects into 1,000 groups, each having similar sums of nonspatial attributes and then selected one object from each group for update. The experiments were conducted on a single core. The resulted minimum, average, and maximum elapsed times are shown in Fig. 13b. From the experiments, we found that only a very small amount of objects affect many others' nondominance scopes. For instance, in whrrl, only three objects can affect over 2,000 objects' nondominance scopes. The majority of objects affect about three or four other objects. As such, the average elapsed time incurred for nondominance scope update is expected to be very close to the minimum elapsed time. The overall average update time is less than 1 second. Despite the maximum elapsed time appears to be very long (e.g., 28 minutes for whrrl), nondominance scope recomputations can be sped up through parallelization.

From this evaluation, we can see that DD outperforms ARTree in support of SQ, RSQ, and TIQ and DD can run in parallel to further boost the search efficiency. ARTree perform faster than DD for SSQ. They both outperform SaLSa and BBS, the state-of-the art approaches. Besides, nondominance scope computation and update can be quickly performed through parallel processing.

6 Conclusion

The notion of locational dominance has been explored to enrich many location-based spatial queries that search for the best nearest spatial objects. Accordingly, several skyline queries, namely, SQ, RSQ, TIQ, and SSQ are formulated and studied. In this paper, we have developed two indexes and corresponding search algorithms based upon different design principles as holistic solutions to efficiently address those queries. In short, we have made four following major contributions:

1. In addition to well-studied SQs, we have formally defined RSQs, TIQs, and SSQs and discussed their applications.
2. We have devised augmented R-tree and corresponding algorithms to support SQs, RSQs, TIQs, and SSQs. Besides, we have introduced the notion of validity distance to determine whether SQs could share an identical result, which speeds up the computation of objects' degrees of influence.
3. We have derived dominance diagram, a composition of (indexed) nondominance scopes. We have detailed the formulation, computation and maintenance of nondominance scopes and corresponding search algorithms to evaluate SQs, RSQs, and TIQs. We have also introduced the notion of nondominance scope envelop to support SSQs.
4. We have conducted comprehensive empirical studies to evaluate our proposed approaches in comparison with the representative state-of-the-art approaches. The experiment results consistently demonstrate the good performance of our approaches. Among our proposed solutions, dominance diagram is the most efficient approach for SQs, RSQs, and TIQs, whereas augmented R-tree is desirable to SSQs.

Thus far, we have considered stationary objects and fixed query points in formulating the indexes supporting studied skyline queries. We shall investigate the possibilities of our proposed approaches for moving objects and moving query points in the future. We also plan to study the issue of uncertain nonspatial attributes and uncertain locations in skyline queries [18].

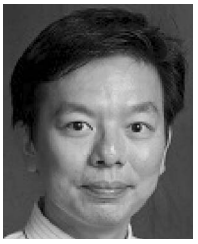
Acknowledgments

In this research, Ken C.K. Lee was in part supported by UMass Dartmouth's Healey Grant 2010-2011. Chi-Yin Chow was partially supported by grant from City University of Hong Kong (Project No. 7002722). Also, this research was completed when Ken C.K. Lee was with UMass Dartmouth.

References

[1] I. Bartolini, P. Ciaccia, and M. Patella, "Efficient Sort-Based Skyline Evaluation," *ACM Trans. Database Systems*, vol. 33, no. 4, pp. 1-49, 2008.
[2] J.L. Bentley, H.T. Kung, M. Schkolnick, and C.D. Thompson, "On the Average Number of Maxima in a Set of Vectors and Applications," *J. ACM*, vol. 25, no. 4, pp. 536-543, 1978.
[3] K.S. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft, "When Is "Nearest Neighbor" Meaningful?" *Proc. Seventh Int'l Conf. Database Theory (ICDT)*, pp. 217-235, 1999.
[4] S. Börzsönyi, D. Kossmann, and K. Stocker, "The Skyline Operator," *Proc. 17th Int'l Conf. Data Eng. (ICDE)*, pp. 421-430, 2001.
[5] D.R. Butenhof, *Programming with POSIX Threads*. Addison Wealey, 1997.
[6] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang, "Skyline with Presorting," *Proc. 19th Int'l Conf. Data Eng. (ICDE)*, pp. 717-816, 2003.
[7] E. Dellis and B. Seeger, "Efficient Computation of Reverse Skyline Queries," *Proc. 33rd Int'l Conf. Very Large Data Bases (VLDB)*, pp. 291-302, 2007.
[8] Y.J. García, M.A. Lopez, and S.T. Leutenegger, "A Greedy Algorithm for Bulk Loading R-Trees," *Proc. Sixth ACM Int'l Symp. Advances in Geographic Information Systems (GIS)*, pp. 163-164, 1998.
[9] G.R. Hjaltason and H. Samet, "Distance Browsing in Spatial Databases," *ACM Trans. Database Systems*, vol. 24, no. 2, pp. 265-318, 1999.
[10] Z. Huang, H. Lu, B.C. Ooi, and A.K.H. Tung, "Continuous Skyline Queries for Moving Objects," *IEEE Trans. Knowledge and Data Eng.*, vol. 18, no. 12, pp. 1645-1658, Dec. 2006.
[11] K. Kodama, Y. Iijima, X. Guo, and Y. Ishikawa, "Skyline Queries Based on User Locations and Preferences for Making Location-Based Recommendations," *Proc. Int'l Workshop Location Based Social Networks (LBSN)*, pp. 9-16, 2009.
[12] F. Korn and S. Muthukrishnan, "Influence Sets Based on Reverse Nearest Neighbor Queries," *Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD)*, pp. 201-212, 2000.

[13] D. Kossmann, F. Ramsak, and S. Rost, "Shooting Stars in the Sky: An Online Algorithm for Skyline Queries," *Proc. 28th Int'l Conf. Very Large Data Bases (VLDB)*, pp. 275-286, 2002.
[14] K.C.K. Lee, "Efficient Evaluation of Location-Dependent Skyline Queries Using Non-Dominance Scopes," *Proc. Second Int'l Conf. Computing for Geospatial Research and Applications (COM.Geo)*, 2011.
[15] K.C.K. Lee, W.-C. Lee, B. Zheng, H. Li, and Y. Tian, "Z-SKY: An Efficient Skyline Query Processing Framework Based on Z-Order," *The VLDB J.*, vol. 19, no. 3, pp. 333-362, 2010.
[16] M.-L. Lee, W. Hsu, C.S. Jensen, B. Cui, and K.L. Teo, "Supporting Frequent Updates in R-Trees: A Bottom-Up Approach," *Proc. 29th Int'l Conf. Very Large Data Bases (VLDB)*, pp. 608-619, 2003.
[17] M.-W. Lee and S. won Hwang, "Continuous Skylining on Volatile Moving Data," *Proc. DBRank: Third International Workshop on Ranking in Databases in conjunction with IEEE International Conference on Data Engineering (ICDE '09)*, pp. 1568-1575, 2009.
[18] X. Lian and L. Chen, "Reverse Skyline Search in Uncertain Databases," *ACM Trans. Database Systems*, vol. 35, no. 1, pp. 1-49, 2010.
[19] B. Liu and C.-Y. Chan, "ZINC: Efficient Indexing for Skyline Computation," *Proc. VLDB Endowment*, vol. 4, no. 3, pp. 197-207, 2010.
[20] M. de Berg, M. van Krefeld, M. Overmars, and O. Schwarzkopf, *Computational Geometry - Algorithms and Applications*. Springer, 2000.
[21] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "An Optimal and Progressive Algorithm for Skyline Queries," *Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD)*, pp. 467-478, 2003.
[22] J.B. Rocha-Junior, A. Vlachou, C. Doulkeridis, and K. Nørvåg, "Efficient Processing of Top-K Spatial Preference Queries," *Proc. VLDB Endowment*, vol. 4, no. 2, pp. 93-104, 2010.
[23] N. Roussopoulos, S. Kelley, and F. Vincent, "Nearest Neighbor Queries," *Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD)*, pp. 71-79, 1995.
[24] M. Sharifzadeh and C. Shahabi, "Spatial Skyline Queries," *Proc. Int'l Conf. Very Large Data Bases (VLDB)*, pp. 751-762, 2006.
[25] K.-L. Tan, P.-K. Eng, and B.C. Ooi, "Efficient Progressive Skyline Computation," *Proc. 27th Int'l Conf. Very Large Data Bases (VLDB)*, pp. 301-310, 2001.
[26] Y. Theodoridis, "R-Tree Portal," http://www.rtreeportal.org, 2013.
[27] L. Tian, L. Wang, P. Zou, Y. Jia, and A. Li, "Continuous Monitoring of Skyline Query over Highly Dynamic Moving Objects," *Proc. Sixth ACM Int'l Workshop Data Eng. for Wireless and Mobile Access (MobiDE)*, pp. 59-66, 2007.
[28] M. Ye, P. Yin, W.-C. Lee, and D.L. Lee, "Exploiting Geographical Influence for Collaborative Point-of-Interest Recommendation," *Proc. 34th Int'l ACM SIGIR Conf. Research and Development in Information Retrieval (SIGIR)*, pp. 325-334, 2011.
[29] B. Zheng, K.C.K. Lee, and W.-C. Lee, "Location-Dependent Skyline Query," *Proc. Ninth Int'l Conf. Mobile Data Management (MDM)*, pp. 148-155, 2008.



Ken C.K. Lee received the PhD degree from the Pennsylvania State University in 2009. He is currently a technical staff in Amazon Web Services (AWS). Prior to joining AWS, he was an assistant professor of computer and information science in the University of Massachusetts Dartmouth. His research interests include cloud computing, intensive data computation, and location-based services.

Baihua Zheng received the PhD degree in computer science from the Hong Kong University of Science and Technology. Currently, she is an associate professor in the School of Information Systems at Singapore Management University. Her research interests include mobile and pervasive computing, and spatial databases. She is a member of the IEEE and ACM.

Cindy Chen received the BS degree in space physics from Peking University, China, the MS and PhD degrees in computer science from the University of California, Los Angeles. Since 2003, she has been a faculty member at the Computer Science Department at the University of Massachusetts Lowell. Her research interests include spatiotemporal databases, data mining, and ontology.

Chi-Yin Chow received the PhD degree from the University of Minnesota-Twin Cities in 2010. He is currently an assistant professor in the Department of Computer Science at the City University of Hong Kong. His research interests include databases, mobile computing, and location-based services. He was an intern at the IBM Thomas J. Watson Research Center during the summer of 2008. He was the coorganizer of ACM SIGSPATIAL MobiGIS 2012. He is a member of the IEEE.