Identifying the Most Endangered Objects from Spatial Datasets

Hua Lu Man Lung Yiu

Department of Computer Science, Aalborg University, Denmark
{luhua,mly}@cs.aau.dk

Abstract. Real-life spatial objects are usually described by their geographic locations (e.g., longitude and latitude), and multiple quality attributes. Conventionally, spatial data are queried by two orthogonal aspects: spatial queries involve geographic locations only; skyline queries are used to retrieve those objects that are not dominated by others on all quality attributes. Specifically, an object p_i is said to dominate another object p_j if p_i is no worse than p_j on all quality attributes and better than p_j on at least one quality attribute. In this paper, we study a novel query that combines both aspects meaningfully. Given two spatial datasets P and S, and a neighborhood distance δ, the *most endangered object query* (MEO) returns the object $s \in S$ such that within the distance δ from s, the number of objects in P that dominate s is maximized. MEO queries appropriately capture the needs that neither spatial queries nor skyline queries alone have addressed. They have various practical applications such as business planning, online war games, and wild animal protection. Nevertheless, the processing of MEO queries is challenging and it cannot be efficiently evaluated by existing solutions. Motivated by this, we propose several algorithms for processing MEO queries, which can be applied in different scenarios where different indexes are available on spatial datasets. Extensive experimental results on both synthetic and real datasets show that our proposed advanced spatial join solution achieves the best performance and it is scalable to large datasets.

1 Introduction

Real-life spatial objects (e.g., hotels) are not only associated with geographic locations but also with multiple quality attributes (e.g., price and star). Conventionally, spatial objects are retrieved by using two orthogonal query types. Spatial queries (e.g., nearest neighbor query, closest pair query) select objects solely based on their geographic locations, or derived measures such as distances. These queries fail to utilize the rich information captured by quality attributes.

On the other hand, the skyline query [1] is a powerful multi-criteria optimization tool for retrieving objects based on their quality attributes. Specifically, this query returns the objects that are not dominated by others on all quality attributes. An object p_i is said to dominate another object p_j if p_i is no worse than p_j on all quality attributes and better than p_j on at least one quality attribute.

Unfortunately, skyline queries focus only on the quality attributes, disregarding the importance of spatial distances among the objects.

Motivation of Retrieving the Most Endangered Objects.
Nowadays, spatial decision support systems need to combine both the location and quality aspects of spatial objects in a meaningful way to retrieve desired objects for the users. This is especially true for a practical application that identifies objects in endangered positions and conducts appropriate planning for them. As an example, suppose that a hotel chain is facing financial challenges and it plans to shut down one of its hotels. Intuitively, a hotel is unlikely to make profit if it is located close to a large number of competitor hotels that have dominating advantages on all quality attributes. Such a hotel may be considered to be shut down. Note that the business of a hotel is not significantly affected by competitor hotels that are far away. Thus, two hotels are considered as geographically close if their distance is within a given neighborhood distance δ.

hotel	price	star
s_1	\$200	4
s_2	\$100	2
s_3	\$250	5
s_4	\$160	3
s_5	\$160	3

Fig. 1. Candidate set S

hotel	price	star
h_1	\$180	4
h_2	\$200	3
h_3	\$200	5
h_4	\$250	3
h_5	\$200	5
h_6	\$220	4
h_7	\$100	2
h_8	\$150	3
h_9	\$200	5
h_{10}	\$160	4

Fig. 2. Competitor set P

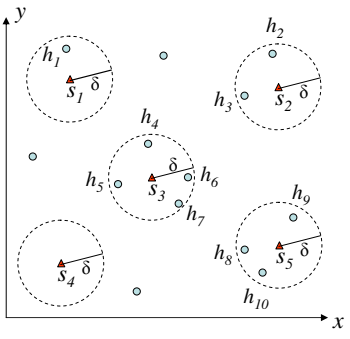


Fig. 3. Locations of hotels

Specifically, given a spatial dataset P (of competitors), a spatial dataset S (of candidates), and a neighborhood distance δ, the *most endangered object query* (MEO) query returns the object $s \in S$ such that within the distance δ from s, the number of objects of P that dominate s is maximized.

Figure 1 lists the quality attributes (price and star) of hotels in a candidate set S that belongs to a hotel chain. Similarly, Figure 2 shows the quality attributes of the set P of competitor hotels. Here, lower prices and higher stars are preferred. For example, the candidate s_1 is dominated by the competitors h_1, h_3, h_5. The locations of all hotels are shown in Figure 3, where competitors are drawn as dots, candidates are shown as triangles, and the neighborhood distance δ is indicated by dashed circles. Within the distance δ, the hotel s_1 is only dominated by h_1. s_2 is dominated by no hotels, even though it is surrounded by h_2 and h_3. Next, s_3 is dominated by the nearby h_5 only; s_5 is dominated by both h_8 and h_{10}. Therefore, the MEO query returns the hotel s_5 as the result since it is dominated by the largest number of competitors within the spatial distance δ.

In addition to the business planning application described above, MEO queries also provide useful results for other fields. For online war games (e.g., War of Warcraft), P is the set of enemy troop locations and S is the set of ally troop

locations. Each troop can be described by multiple quality attributes like solider number, equipment level, etc. The MEO query can then be used to identify the most endangered troop of S that need additional combat support. In wild animal protection, S denotes the set of endangered species and P represents their enemies. Quality attributes refer to abilities such as strength, agility, and stamina. The MEO query helps to identify the animal most worthy of extra protection.

Deficiency of Existing Solutions.
It is noteworthy that neither a conventional spatial query nor a skyline query alone will return the same result as a MEO query. A skyline query on the competitor set followed by a spatial query with respect to all candidates does not help either, because some candidates are dominated by non-skyline competitors. Referring to our earlier example, the candidate s_1 is dominated by the competitor h_1 which however is not a skyline object among the competitors (since h_1 is dominated by h_{10}).

In Section 2.3, we will elaborate two straightforward solutions that correctly process the MEO query: (i) an RDBMS solution, and (ii) a multi-step R-tree based solution. Those solutions do not fully exploit the characteristics of the MEO query so they incur high processing cost.

Our Contributions.
Motivated by these observations, we propose several algorithms for processing MEO queries. These algorithms can be applied in different scenarios where different indexes are available on input datasets. Our first algorithm is a baseline iterative approach which needs only an R-tree index on the dataset P. Our next two algorithms require that the dataset S is indexed by an R-tree R_S, and the dataset P is indexed by an aggregate R-tree R_P. In the aggregate R-tree [12], each node entry stores a count of objects in its subtree. Then the query is processed by a depth-first or best-first search on the tree R_S, based on efficient upper bound counting techniques using R_P. Our last algorithm requires the same indexes as the previous two, but it evaluates the query in a spatial join manner.

The remainder of this paper is organized as follows. Section 2 formally defines the MEO query and briefly reviews the related work. Section 3 develops our algorithms for processing MEO queries. Section 4 presents extensive experimental results of our proposals on both synthetic and real data. Finally, Section 5 concludes the paper and discusses future research directions.

2 Preliminaries

2.1 Problem Statement

We assume that all quality attributes are numeric and each attribute domain is totally ordered. Let c be the number of quality attributes. A *quality vector* is a point ψ in the c-dimensional space $\mathbb{R}^c$, where each dimension refers to a quality attribute. $\psi[i]$ denotes the i-th (quality) attribute value of ψ.

Without the loss of generality, we assume that smaller values are preferred to larger ones in quality attributes throughout this paper. According to [1], a quality vector ψ *dominates* another one ψ' (denoted as $\psi \prec \psi'$) when:

$$(\exists\ 1 \leq i \leq c,\ \psi[i] < \psi'[i]) \wedge (\forall\ 1 \leq i \leq c,\ \psi[i] \leq \psi'[i]) \tag{1}$$

A *location* is a pair (x, y) in the Euclidean space $\mathbb{R}^2$, where x and y are coordinate values. A *spatial object* $o = \langle loc, \psi \rangle$ consists of both a location $o.loc$ and a quality vector $o.\psi$. The notation $dist(o, o')$ denotes the Euclidean distance between the locations of the spatial objects o and o'. Given two spatial objects o and o', o is said to *dominate* o' when $o.\psi \prec o'.\psi$.

Definition 1. (Neighborhood Dominating Score) *Given a spatial object set P, a spatial object s, and a neighborhood distance δ, the neighborhood dominating score of s on P with respect to δ is defined as:*

$$\Phi_{P,\delta}(s) = |\{o \in P \mid dist(o, s) \leq \delta, o.\psi \prec s.\psi\}|$$

Whenever the context becomes clear, we drop the subscripts of $\Phi(s)$. We then define the *most endangered object query* (MEO) as follows. Our objective is to design an I/O-efficient solution for processing MEO queries on large datasets.

Definition 2. (Most Endangered Object Query) *Given two spatial object sets P and S, for competitors and candidates respectively, and a neighborhood distance δ, the* **most endangered object query** *(MEO) returns from S an object s such that $\Phi_{P,\delta}(s)$ is maximized, i.e., $\forall\ s' \in S,\ \Phi_{P,\delta}(s) \geq \Phi_{P,\delta}(s')$*

2.2 Related Work

Spatial Join.
Given a distance bound δ, and two spatial datasets S and P, *the δ-distance join* retrieves each pair $\langle s, p \rangle$ (where $s \in S$ and $p \in P$) such that their Euclidean distance $dist(s, p)$ is within δ. The *R-tree join* (RJ) [2] can be applied to evaluate the δ-distance join if both S and P are indexed by R-trees R_S and R_P respectively. RJ first examines the entries in the root nodes of R_S and R_P. If an entry e_S (of the tree R_S) and an entry e_P (of the tree R_P) satisfies $mindist(e_S, e_P) \leq \delta$, then the subtrees of e_S and e_P may contain some objects within δ. In that case, RJ is recursively applied on the subtrees of e_S and e_P. Eventually, RJ reaches the leaf level and reports the pairs of objects that are within δ. Efficient δ-distance join algorithms on high-dimensional data have been studied in [8]. Zhu et al. [19] proposed the *top-k spatial join* for computing k objects of S that intersect with the largest number of objects in P.

The above studies consider only the spatial relationship between the objects in S and P, but not their dominance relationship on quality attributes of objects in our MEO query.

Location Selection Queries.
In the literature, various constraints have been combined with conventional spatial queries in order to select semantically optimal locations or objects. Du et

al. [5] proposed the optimal-location query. Given a site set S, a weighted object set O, and a spatial region Q, the optimal-location query returns a location in Q with the maximum influence. The influence of a location l is defined as the total weights of objects in O, each of which has l as its nearest neighbor in the set $S \cup \{l\}$. Using the same influence definition, Xia et al. [15] formulated a different top-k most influential spatial sites query, which returns k sites (from S and within Q) having the highest influences. In the same context, Zhang et al. [17] proposed the min-dist optimal-location query. Rather than maximizing influence, this query selects from Q a location l which minimizes the average distance from every object in O to its nearest site in $S \cup \{l\}$. All these optimal-location queries differ from our MEO query in the sense that they do not consider multi-dimensional dominance relationship among the non-spatial quality attributes of the objects. This renders their solutions inapplicable to our problem.

Yiu et al. [16] formalized the top-k spatial preference query, which returns the k spatial objects with the highest ranking scores, based on the feature qualities in their spatial proximity. Such score functions, however, do not support multi-dimensional dominance relationship as in our MEO query.

Li et al. [9] proposed Dominant Relationship Analysis (DRA), for discovering the dominant relationship between products and potential consumers. To efficiently answer different analysis queries of DRA, the authors proposed a datacube structure, named *DADA*, which stores the dominant relationships in the way supporting ordered access and compressing. Li et al. [10] combined dominance relationship with spatial distance and defined complex location selection problems. However, its solution cannot be applied to solve our problem. Only one dataset is considered in [10], from which desirable objects are selected. In contrast, our problem involves two datasets with different practical semantics, and aims at selecting an object (from the candidate dataset S) with the highest score defined with respect to the competitor dataset P.

Skyline Queries in Spatial and Spatiotemporal Settings.
Skyline queries has been adopted in spatial and spatiotemporal database to define specific problems. Huang and Jensen [6] proposed an in-route skyline query for location-based services. When moving along a pre-defined road route towards her/his destination, a user may visit points of interest in the network. Points to visit are selected in terms of multiple distance-related preferences like detour and total traveling distance. The authors optimize such selections using skyline queries involving specific interesting dimensions.

Sharifzadeh and Shahabi [14] studied the spatial skyline query, which is in fact a specialized version of the dynamic skyline query [13]. Given a set of query points $Q = \{q_1, \ldots, q_n\}$ and two points p and p', p is said to spatially dominate p' iff $dist(p, q_i) \leq dist(p', q_i)$ for any $q_i \in Q$ and $dist(p, q_i) < dist(p', q_i)$ for at least one $q_i \in Q$. The spatial skyline of a set of points P is the subset of all points not spatially dominated by any other point of P. Observe that such queries consider only spatial attributes but not any non-spatial quality attributes.

Huang et al. [7] defined continuous skyline query in a spatiotemporal context. A spatial object p dominates another object p' with respect to a query location

q, if p is closer to q than p' and p dominates p' on all non-spatial attributes. A continuous skyline query then maintains all spatial objects not dominated by any others, while the query q is continuously moving along a specified trajectory in the Euclidean space. Using a similar setting, Zheng et al. [18] addressed how to compute the valid scope for such a query result without knowing the movement pattern of the object.

Our MEO query is also different from the constrained skyline query [13], in which the objects being considered is restricted by a given constraint region in the domain space of (multiple) quality attributes. In contrast, the spatial distance constraint δ employed in the MEO query is only used in the spatial domain but not on quality attributes.

The main difference of the MEO query from the above studies is that the MEO query is not a skyline problem. Recall from the motivation example in the Introduction that a non-skyline object (e.g., hotel h_1) in the competitor set can still dominate a candidate object (e.g., hotel s_1) within its spatial neighborhood.

2.3 Straightforward Solutions

In this section, we describe two straightforward solutions for evaluating the MEO query, and then discuss their drawbacks.

RDBMS Solution (SQL).
In fact, the MEO query can be expressed by the following SQL statement (see Figure 4), and thus it can be executed in any existing commercial RDBMS. Here, we assume that the input datasets are stored in two relational tables `S` and `P`, which share the same schema. The attribute `id` is the identifier of a tuple. The attributes `x` and `y` represent spatial coordinates; whereas the attributes `psi_1`, `psi_2`, etc, are the quality attributes. The query parameter δ is translated to the value `delta` in the SQL query.

In the following query, we first join the tuples of the tables `S` and `P`. Definition 1 is expressed by the join condition in the WHERE clause: the first line refers to the neighborhood distance constraint, and the last two lines represent the dominance comparison. After that, the intermediate join result set is partitioned into groups based on its `id` in `S`. Then, the count of each group is computed and the `id` of the largest group (together with its count) is returned as the result.

```
SELECT      S.id, COUNT(*)
FROM        S, P
WHERE       (S.x-P.x)*(S.x-P.x)+(S.y-P.y)*(S.y-P.y)<=delta*delta
            AND ( P.psi_1<=S.psi_1 AND P.psi_2<=S.psi_2 AND ... )
            AND ( P.psi_1< S.psi_1 OR  P.psi_2< S.psi_2 OR  ... )
GROUP BY    S.id
ORDER BY    COUNT(*)       DESC LIMIT 1
```

Fig. 4. Expression of MEO Query in SQL

The main disadvantage of this method is that it incurs very high execution time in existing RDBMS. Even though typical indexes (e.g., B^+-trees and hash

indexes) may be used by the RDBMS engine, it cannot fully exploit the complex join condition (shown in the WHERE clause) for optimizing the search cost.

Multi-step R-tree based Solution (3Step).
Another straightforward method for the MEO query is as follows, assuming that the datasets S and P are indexed by two R-trees R_S and R_P respectively.

In the first step, this method performs the δ-distance join by applying the RJ algorithm [2] on those two trees, in order to obtain the pairs $\langle s, p\rangle$ that are within the δ distance. In the second step, any pair $\langle s, p\rangle$ is pruned if it does not satisfy $p.\psi \prec s.\psi$. In the third step, the remaining pairs are assigned into groups according to $s.id$, and the object s having the largest group count is reported as the result.

The drawback of this method is that the first step incurs a high cost at a large δ value, regardless of the pruning effectiveness of the second step.

3 Algorithms for Most Endangered Object Queries

We in this section detail our algorithms for processing most endangered object queries. We first present the baseline approach that carries out an iterative search on all candidate objects without any index on them. Then improved algorithms are presented with specific index requirements. Table 1 lists the notations to be used throughout the paper.

Notation	Meaning
P	the set of objects for competitors
S	the set of objects for candidates
$\psi \prec \psi'$	a quality vector ψ dominates another one ψ'
$dist(o, o')$	Euclidean distance between two objects o and o'
$mindist(e, e')$	minimum distance between two R-tree entries e and e'
$\Phi(s)$	neighborhood dominating score of an object s
$\odot(s, \delta)$	a circular region with center s and radius δ
$\Xi(e, \delta)$	δ-Minkowski region of an R-tree entry e

Table 1. Table of Notations

3.1 A Baseline Approach: Iterative Search Algorithm

In this section, we assume that the dataset P is indexed by an R-tree R_P and the dataset S is not indexed. We first present a basic algorithm for computing the score $\Phi(s)$ of an object $s \in S$, and then apply it iteratively on each object in order to obtain the final result.

ObjectScore (see Algorithm 1) is a recursive algorithm for computing the $\Phi(s)$ value of the object s with respect to the objects in the subtree of the entry e_P (of the R-tree R_P). The input parameter δ represents the distance threshold. At line 1, the counter v is used to maintain the value of $\Phi(s)$. In case e_P is a leaf entry (line 2), we check whether its distance to s is within δ and its quality vector dominates that of s. If so, then the counter v is incremented. When e_P is a non-leaf entry (line 5), we read its the child node, and recursively process each

of its entry e'_P if the minimum distance $mindist(e'_P, s)$ from e'_P to s is within δ.

Algorithm 1 ObjectScore(Object s, Entry e_P of the R-tree R_P, Distance δ)

```
1: v := 0
2: if e_P is a leaf entry then
3:     if dist(e_P, s) ≤ δ and e_P.ψ ≺ s.ψ then
4:         v := 1
5: else                                          ▷ e_P is a non-leaf entry
6:     read the child node CN pointed to by e_P;
7:     for each entry e'_P in CN do
8:         if mindist(e'_P, s) ≤ δ then
9:             v := v+ObjectScore(s, e'_P, δ)
10: return v
```

We then propose the iterative search (**IS**) for processing the MEO query. Its pseudo code is shown in Algorithm 2. It takes as input (i) an R-tree on competitor set P, (ii) the candidate object set S, and (iii) the distance δ. The object *meo* is used to keep track the result object found so far, and the value γ corresponds to the score of *meo*. At line 1, we initialize *meo* to null and γ to 0 respectively. For each location s of the set S, the algorithm applies the **ObjectScore** function on the root of R-tree R_P (lines 2–3) to obtain the score $\Phi(s)$ of s. If $\Phi(s)$ is higher than γ, then the result and its score will be updated (lines 4–6). Finally, the algorithm returns the object *meo* as the result.

The IS algorithm is able to exploit the main-memory buffer better if it processes all the locations of S via a locality-preserving order. Thus, we develop the algorithm IS-Hil, which first applies external sorting on the locations in S based on the Hilbert ordering [4, 11], and then processes them by IS.

Algorithm 2 IS(R-tree R_P on P, Object set S, Distance δ)

```
1: meo :=null; γ := 0
2: for each object s ∈ S do
3:     Φ(s) :=ObjectScore(s, R_P.root, δ)
4:     if Φ(s) > γ then
5:         γ := Φ(s)
6:         meo := s
7: return meo
```

3.2 Aggregate R-tree Search Algorithms

Observe that IS algorithm processes every object once in the set S. We now propose to index the set S by an R-tree R_S and develop an efficient method to prune unqualified subtrees of R_S that cannot contribute to the result.

In order to support efficient counting operations, we index the dataset P by an aggregate `COUNT` R-tree R_P [12]. Specifically, each non-leaf entry e_P of the tree R_P stores an additional *count* value, denoted as $e_P.count$, which is equal to the number of objects in the subtree of e_P.

Derivation of Upper Bound Score.
Suppose that e_S is a non-leaf entry of the tree R_S. Figure 5 shows the spatial

extent of e_S as a rectangular region. We wish to derive an upper bound score $\Phi^+(e_S)$ of e_S such that $\Phi(s) \leq \Phi^+(e_S)$ for any object s in the subtree of e_S.

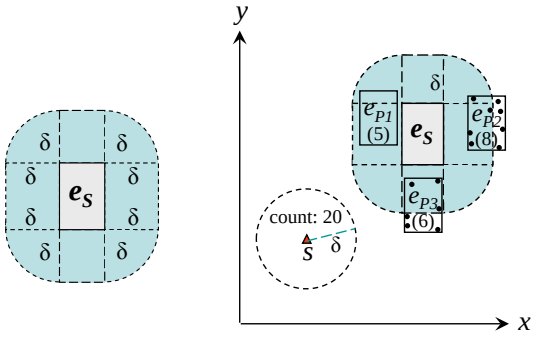


Fig. 5. $\Xi(e_S, \delta)$

Fig. 6. Pruning Rule 1

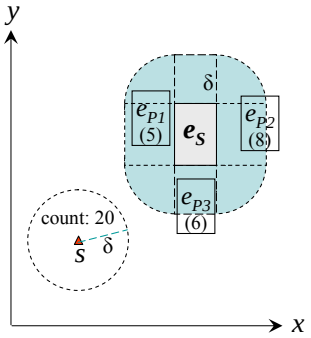


Fig. 7. Pruning using $\Phi^*(e_S)$

First, we introduce the concept of δ-Minkowski region [3] of e_S, denoted by $\Xi(e_S, \delta)$, which is the set of possible locations whose minimum distance from e_S is within the distance δ.

$$\Xi(e_S, \delta) = \{t \in \mathbb{R}^2 \mid mindist(t, e_S) \leq \delta\} \tag{2}$$

The region $\Xi(e_S, \delta)$ is illustrated in Figure 5 as the region extended from the rectangle e_S by the distance δ. Given the dataset P and the distance δ, we define the *upper bound neighborhood dominating score* $\Phi^+_{P,\delta}(e_S)$ of e_S as the number of objects in P that fall into the region $\Xi(e_S, \delta)$.

$$\Phi^+_{P,\delta}(e_S) = |\{o \in P \mid o \subseteq \Xi(e_S, \delta)\}| \tag{3}$$

The nice property of the upper bound score $\Phi^+_{P,\delta}(e_S)$ (of a non-leaf entry e_S) is that it is guaranteed to be greater than or equal to the score $\Phi_{P,\delta}(s)$ of any object s in the subtree of e_S. This is shown in the following lemma.

Lemma 1. *Let δ be a distance threshold and P be a dataset of objects. Given a rectangle e_S, it holds that $\Phi_{P,\delta}(s) \leq \Phi^+_{P,\delta}(e_S)$ for any object s that falls into e_S.*

Proof. Let s be an object that falls into e_S. According to Definition 1, each object $o \in P$ that contributes to $\Phi_{P,\delta}(s)$ must satisfy the inequality $dist(o, s) \leq \delta$ (and also the condition $o.\psi \prec s.\psi$). Each such object o also satisfies $mindist(o, e_S) \leq \delta$ because s falls into e_S. That means such object o falls into the region $\Xi(e_S, \delta)$ and thus contributes to $\Phi^+_{P,\delta}(e_S)$. Therefore, we have $\Phi_{P,\delta}(s) \leq \Phi^+_{P,\delta}(e_S)$. ■

We proceed to present the **EntryScore** algorithm, whose pseudo code is shown in Algorithm 3. It takes as input (i) an entry e_S of the R-tree R_S on S, (ii) an entry e_P in the aggregate R-tree R_P on P, and (iii) the distance threshold δ. This algorithm serves for two purposes, depending on whether e_S is a leaf entry or not. If e_S is a leaf entry, then the algorithm calls the ObjectScore function to compute the exact score $\Phi(e_S)$ of e_S (lines 2–3).

Otherwise, e_S is a non-leaf entry, lines 4–11 are used to compute the upper bound score $\Phi^+(e_S)$ of e_S. We then check whether the region $\Xi(e_S, \delta)$ contains e_P. If so, then each object in the subtree of e_P is guaranteed to fall in $\Xi(e_S, \delta)$. Thus, we increment the counter v by the count $e_P.count$, without visiting the subtree of e_P. If not, then we need to read the child node of e_P. An entry e'_P in the child node is recursively processed if it intersects $\Xi(e_S, \delta)$, i.e., having the potential of contributing to $\Phi^+(e_S)$.

Algorithm 3 EntryScore(Entry e_S of the R-tree R_S on S, Entry e_P in the aggregate R-tree R_P on P, Distance δ)

```
1: v := 0
2: if e_S is a leaf entry then
3:     v :=ObjectScore(e_S, e_P, δ)
4: else                                          ▷ e_S is a non-leaf entry
5:     if Ξ(e_S, δ) contains e_P then
6:         v := e_P.count
7:     else
8:         read the child node CN pointed to by e_P
9:         for each child e'_P in CN do
10:            if Ξ(e_S, δ) intersects e'_P then
11:                v := v+EntryScore(e_S, e'_P, δ)
12: return v
```

Figure 6 illustrates an example of computing the upper bound score $\Phi^+(e_S)$ of a non-leaf entry e_S (of the tree R_S), by using the EntryScore algorithm. Here, the aggregate R-tree R_P (of the dataset P) only has the non-leaf entries e_{P1}, e_{P2}, e_{P3}, whose associated count values are 5, 8, 6 respectively. Since e_{P1} is contained by $\Xi(e_S, \delta)$, its count (5) is added to the upper bound score of e_S, without visiting the subtree. As the entries e_{P2} and e_{P3} intersect $\Xi(e_S, \delta)$, their child nodes need to be accessed. Then, the child nodes of e_{P2} and e_{P3} are found to have 4 and 3 objects, respectively, that fall into the region $\Xi(e_S, \delta)$. Therefore, the values 4 and 3 are added to the upper bound score of e_S. In summary, we obtain: $\Phi^+(e_S) = 5 + 4 + 3 = 12$.

Search Algorithm.

Recall that we have studied the notion of $\Phi^+(e_S)$ (for a non-leaf entry e_S), and the EntryScore algorithm for computing it. We continue to present a pruning rule for reducing the search space, and then develop two algorithms for solving MEO based on the pruning rule.

According to Lemma 1, we devise the following pruning rule to identify an unpromising entry e_S (of the R-tree R_S on S) whose subtree cannot contain the MEO object.

Pruning Rule 1 *Let $s \in S$ be an object from S, and e_S be a non-leaf entry from R-tree R_S on S. If $\Phi(s) > \Phi^+(e_S)$, then the entry e_S can be safely pruned.*

We continue with the example of Figure 6 to illustrate this pruning rule. Suppose that we have already examined object s and computed its exact value

$\Phi(s) = 20$ (by the EntryScore algorithm). Next, we want to check whether it is necessary to visit the subtree of the non-leaf entry e_S. Its upper bound score $\Phi^+(e_S) = 5 + 4 + 3 = 12$ can be computed by the EntryScore algorithm, as discussed before. Since $\Phi^+(e_S) < \Phi(s)$, the entry e_S cannot contribute to the result and therefore it can be safely pruned.

It is desirable to find early an object s with high $\Phi(s)$ value such that unqualified subtrees of R_S can be effectively pruned. The search on R_S can be conducted in two tree search paradigms, namely *best-first search* or *depth-first search*.

The pseudo code of the *best-first search* (BFS) is shown in Algorithm 4. It employs a max-heap H so as to visit the tree entries of R_S in descending order of their upper bound scores. Initially, the algorithm inserts into H the root entry of R_S together with its upper bound score $|P|$. Each time a non-leaf entry is deheaped, all its child entries are enheaped with their own priorities obtained by calling the EntryScore algorithm (lines 5–8). If the entry being deheaped is a leaf entry, it will be returned as the most endangered object (line 9–10). The correctness of the BFS algorithm is guaranteed by (i) the property of the max-heap, and (ii) the upper bound computed by EntryScore($R_P.root, e_S, \delta$) (stated in Lemma 1).

Algorithm 4 BFS(Aggregate R-tree R_P on P, R-tree R_S of S, Distance δ)

```
1: initialize a max-heap H
2: enheap(H, ⟨R_S.root, |P|⟩)
3: while H is not empty do
4:     e_S :=deheap(H)
5:     if e_S is a non-leaf entry then
6:         read the child node CN pointed to by e_S;
7:         for each child e'_S of e_S do
8:             enheap(H, ⟨e'_S, EntryScore(e'_S, R_P.root, δ)⟩)
9:     else
10:        return e_S
```

Similarly, it is also possible to traverse the R-tree R_S in the depth-first manner. The resulting algorithm is called *the depth-first search* (DFS). Due to the space limit, we omit its pseudo code here.

3.3 Spatial Join Based Algorithm

As in Section 3.2, here we assume that the set of candidates objects S is indexed by an R-tree R_S and the set of competitor objects P is indexed by an aggregate R-tree R_P. Recall that both the BFS and DFS algorithms need to compute the upper bound score of a non-leaf entry e_S (of the tree R_S) explicitly by accessing the tree R_P, incurring considerable cost. This section presents a more efficient solution by deriving an upper bound score of e_S with low cost and tightening the score bound gradually whenever necessary.

Formulation of a Join List.
Before proposing the solution, we first introduce several relevant concepts. Let

e_S be an entry of the R-tree R_S. At query time, we associate each encountered entry e_S with its *join list* $e_S.JL$, for storing the entries of the R-tree R_P that may combine with the subtree of e_S to generate potential results.

Specifically, a join list $e_S.JL$ is required to satisfy both of these conditions:

- (i) each entry e_P in $e_S.JL$ satisfies $e_P \cap \Xi(e_S, \delta) \neq \emptyset$,
- (ii) for each $p \in P$ satisfying $p \cap \Xi(e_S, \delta) \neq \emptyset$, there is exactly one ancestor entry e_P (of p) in $e_S.JL$.

The first condition ensures that the entries stored in $e_S.JL$ are relevant to e_S because they intersect the Minkowski region $\Xi(e_S, \delta)$ of e_S. The second condition ensures that there is no missing entry or redundant entry in $e_S.JL$.

The next question is how to check whether a particular join list satisfies both conditions (i) and (ii) stated above. First of all, we start with the root join list $e_S.JL = \{R_P.root\}$, which trivially satisfies the condition (ii). The condition (i) can be easily checked on $e_S.JL$. In each subsequent step, we can apply the following *expansion operation* on $e_S.JL$; this operation guarantees that its output join list must satisfy both conditions (i) and (ii). Each time, we pick an non-leaf entry e_P from $e_S.JL$, read the child node CN pointed to by e_P, and then insert each entry $e'_P \in CN$ satisfying $e'_P \cap \Xi(e_S, \delta) \neq \emptyset$ into the list $e_S.JL$.

Having described the concept of a join list $e_S.JL$, we then define the upper bound score of e_S with respect to $e_S.JL$ as:

$$\Phi^*_{P,\delta}(e_S) = \sum_{e' \in e_S.JL} e'.count \tag{4}$$

The above upper bound score $\Phi^*_{P,\delta}(e_S)$ is guaranteed to be greater than or equal to the score $\Phi_{P,\delta}(s)$ of any object s in the subtree of e_S. This is formally stated in the following lemma.

Lemma 2. *$\Phi^*_{P,\delta}(e_S) \geq \Phi_{P,\delta}(s)$ for any object s that falls into e_S.*

Proof. Let s be an object that falls into e_S. According to Lemma 1, we obtain $\Phi^+_{P,\delta}(e_s) \geq \Phi_{P,\delta}(s)$. From the property (ii) of the join list, we derive $\Phi^*_{P,\delta}(e_S) \geq \Phi^+_{P,\delta}(e_S)$. By combining both inequalities above, we have $\Phi^*_{P,\delta}(e_S) \geq \Phi_{P,\delta}(s)$. ■

We illustrate an example on exploiting the upper bound score of join list for pruning unnecessary subtrees of the tree R_S. Figure 7 shows a non-leaf entry e_S. Suppose that we have encountered an object with $\Phi_{P,\delta}(s) = 20$. Next, we check whether it is necessary to access the child node of e_S. Suppose that $e_S.JL = \{e_{p1}, e_{p2}, e_{p3}\}$, and $\Phi^*_{P,\delta}(e_S) = 5 + 8 + 6 = 19 < 20$, i.e., lower than the score of object s. Therefore, the entry e_S (together with its join list) can be safely pruned as no object in e_S can have higher score than s.

Search Algorithm.
Algorithm 5 is the pseudo code of the spatial join based algorithm. It employs a max-heap H to keep all R_S entries to be processed. Each R_S entry e_S is enheaped together with its join list $e_S.JL$ and a count value obtained from Equation 4.

If the R_S entry e_S being deheaped is a leaf node and its join list is null, it is returned as the most endangered object according to the max-heap property (lines 5–8). If the leaf entry e_S's join list is not null, its exact neighbor dominator count is calculated by calling the ObjectScore algorithm for each entry in its join list (lines 10–12). After that, e_S is enheaped again with a null join list and the calculated count value (line 13).

Algorithm 5 SJB(Aggregate R-tree R_P of P, R-tree R_S of S, Distance δ)

```
 1: initialize a max-heap H
 2: e_root := R_S.root; e_root.JL := {R_P.root}
 3: enheap(H, ⟨e_root, e_root.JL, 0⟩)
 4: while H is not empty do
 5:     ⟨e_S, e_S.JL⟩ := deheap(H)
 6:     if e_S is a leaf entry then
 7:         if e_S.JL is null then
 8:             return e_S
 9:         else
10:             v := 0
11:             for each e_j in e_S.JL do
12:                 v := v+ObjectScore(e_S, e_j, δ)
13:             enheap(H, ⟨e_S, null, v⟩)
14:     else
15:         read the child node CN_S pointed to by e_S
16:         for each entry e_i in CN_S do
17:             v := 0; e_i.JL := ∅
18:             for each e_j in e_S.JL do
19:                 if Ξ(e_i, δ) contains e_j then
20:                     add e_j to e_i.JL;  v := v + e_j.count
21:                 else
22:                     read the child node CN_P pointed to by e_j
23:                     for each child e' in CN_P do
24:                         if Ξ(e_i, δ) intersects e' then
25:                             add e' to e_i.JL;  v := v + e'.count
26:             enheap(H, ⟨e_i, e_i.JL, v⟩)
```

Otherwise, the join is executed by expanding the non-leaf R_S entry e_S being deheaped, and enheaping each subentry in e_S with its corresponding join list and count value (15–26). In particular, when e_S is expanded its each subentry e_i gets part of entries in $e_S.JL$ as $e_i.JL$. In this way, as the join proceeds on the R_S entries are enheaped with join lists of smaller coverage, thus giving tighter upper bounds of neighbor dominator counts which favors pruning.

4 Experimental Study

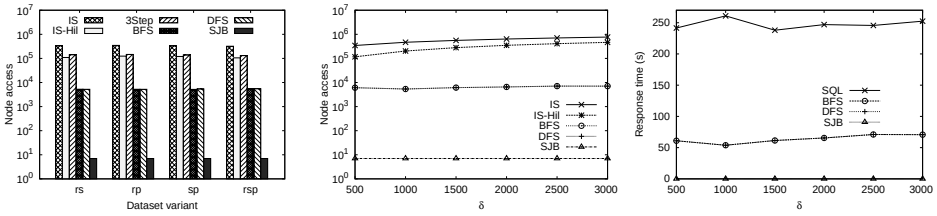
In this section, we experimentally evaluate our proposed algorithms for processing MEO queries. All algorithms were implemented in Java and were run on a Windows XP PC with a 2.8GHz Intel Pentium D CPU and 1GB RAM. We used synthetic and real datasets for both the competitor set P and the candidate set S. In each dataset, all spatial coordinates are normalized to Euclidean space

$[0,10000]^2$, whereas each quality attribute is normalized to the unit interval [0,1]. In all experiments, the disk page size is set to 4KBytes and a LRU memory buffer with 512KBytes is used. We issue 20 queries for each test case in each experiment, and the spatial distance constraint δ in each query is a random value in (0,1000]. We measure the average node access cost per each query because it dominates the total query processing cost.

4.1 Experimental Results on Real Data

In this section, we used two real datasets from AllStays.com[1] that maintains collections of hotels, resorts, campgrounds, etc. around the world. We chose the dataset of hotels in US and cleaned it up as follows. We removed all those records without longitude and latitude, and discarded quality attributes with very few non-null values. For each remaining quality attribute, any null value was replaced by a random value from its attribute domain. As a result, we obtained 30,918 hotel records with the schema (*longitude*, *latitude*, *review*, *stars*, *price*). Value conversion was done on a quality attribute if necessary, e.g., a higher stars value was converted to a lower value in the normalized range [0, 1]. This way, lower values are preferable to higher ones. After normalizing the hotel records as mentioned above, one third (10,306 records) are randomly picked as the S dataset and the others (20,612 records) form the P dataset.

We then used different quality attribute combinations and got four variants of P dataset: *review* and *stars* (denoted as rs), *review* and *price* (denoted as rp), *stars* and *price* (denoted as sp), and all three attributes (denoted as rsp). The corresponding S dataset variants were obtained in the same way. We then performed three groups of experiments on the real datasets obtained.



(a) Node access, all datasets (b) Node access vs. δ, on rsp (c) Response time, on rsp

Fig. 8. Results on real datasets

The first group of experiments investigated into the effect of different real dataset variants. We also implemented the multi-step R-tree based solution mentioned in Section 2.3 (named 3Step for short). The average node access cost results are reported in Figure 8(a). The 3Step solution is inefficient because it cannot exploit the characteristics of the MEO query for effective pruning. The SJB algorithm has the lowest cost.

The second group of experiments studied the effect of distance constraint δ. We used the rsp datasets and varied δ from 500 to 3000. The results are reported

[1] Hotel and Travel Guide. http://www.allstays.com/

in Figure 8(b). Since SJB applies an effective pruning technique, it outperforms the other algorithms.

In the last group of experiments, we executed the RDBMS solution mentioned in Section 2.3 (named SQL for short) on the rsp dataset. We executed SQL in the Oracle RDBMS; we could only obtain the response time of SQL but not its I/O cost. Therefore, we compared the response time of SQL with our proposed BFS, DFS and SJB algorithms (see the results in Figure 8(c)). Obviously, our algorithms incur considerably shorter response time than SQL.

In subsequent experiments, we discard the solutions 3Step and SQL due to their high query cost.

4.2 Scalability and Robustness Experiments on Synthetic Data

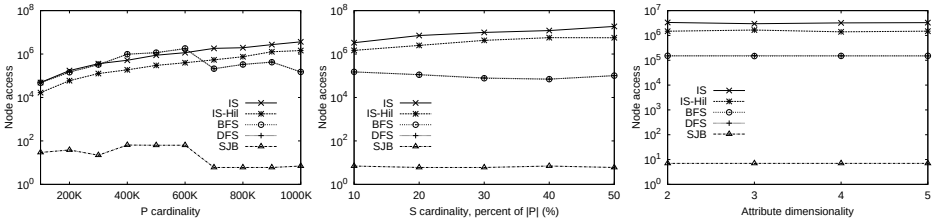
Having studied the performance of our algorithms on real data, we now test their scalability and robustness by using synthetic data. Table 2 lists the parameters for the generation of synthetic datasets; the default parameter values are shown in bold. The cardinality of the set P varies from 100K to 1000K. For each P set, the cardinality of the corresponding S set changes from 10% to 50% of the cardinality of P. For both P sets and S sets, all locations are generated randomly in the normalized Euclidean space $[0,10000]^2$. The quality attribute dimensionality of those datasets varies from 2 to 5. For both P and S sets, we generated quality values following the independent (IN) distribution and the anti-correlated (AC) distribution, according to Borzonyi et al. [1].

Parameter	**Setting**
Competitor dataset cardinality, $\|P\|$	100K, 200K, ..., **1000K**
Candidate dataset cardinality, $\|S\|$	**10%**·$\|P\|$, 20%·$\|P\|$, ..., 60%·$\|P\|$
Quality attribute dimensionality, c	**2**, 3, 4, 5
Quality attribute distribution	Independent (IN), Anti-correlated (AC)

Table 2. Parameters of synthetic datasets

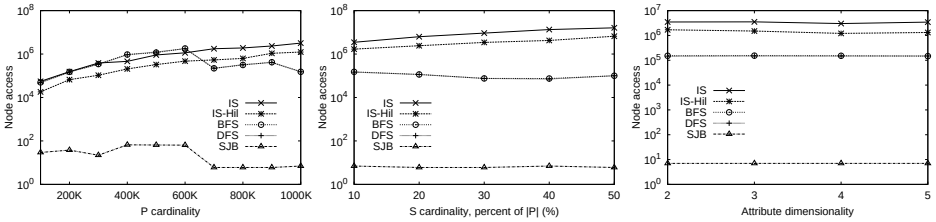
Effect of Competitor Dataset Cardinality $|P|$.
We first varied $|P|$ from 100K to 1000K to see its effect on the performance of all algorithms. The results are reported in Figure 9(a) and Figure 10(a), on IN and AC distribution respectively. As $|P|$ increases, both IS and IS-Hil algorithms incur more node accesses. The reason is that the recursive ObjectScore algorithm needs to access more nodes from the R-tree on a larger P. While BFS and DFS algorithms degrade first and then improve as $|P|$ increases. As $|S|$ was fixed to 10% of that of $|P|$, a larger P leads to a larger S and a larger R-tree on S. Therefore BFS and DFS degrade as they have to search a larger R-tree. The subsequent improvement is resulted from a better organized R-tree on S, which is achieved only when S contains enough objects. The performance of the SJB algorithm fluctuates more visibly, as the join operation complicates the use of both the R-tree on S and the aggregate R-tree on P. Among all algorithms, SJB performs the best as the join is very efficient due to the powerful pruning rule it employs.

Effect of Candidate Dataset Cardinality $|S|$.
We then varied $|S|$ from 10% to 50% of that of $|P|$. The results on the effect



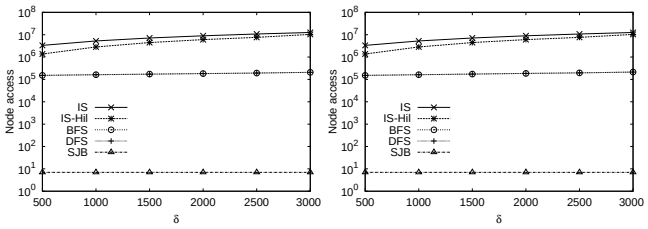
(a) Effect of $|P|$ (b) Effect of $|S|$ (c) Effect of c

Fig. 9. Node access on synthetic datasets, with IN quality attributes



(a) Effect of $|P|$ (b) Effect of $|S|$ (c) Effect of c

Fig. 10. Node access on synthetic datasets, with AC quality attributes



(a) IN quality attributes (b) AC quality attributes

Fig. 11. Node access vs. δ

of varying $|S|$ are reported in Figure 9(b) and Figure 10(b). Observe that the SJB algorithm still outperforms all others; BFS and DFS are the second best among all. As $|S|$ increases, both BFS and DFS improve slightly. This again is attributed to a better organized R-tree on S.

Both IS and IS-Hil algorithms degrades slightly as $|S|$ increases, because they invoke the recursive ObjectScore algorithm for every object in S. Whereas the latter performs slightly better steadily as $|S|$ varies. Accessing objects of S in the Hilbert curve order enables IS-Hil to considerably reuse R_P tree nodes in the buffer, which offsets the effect of increasing $|S|$.

Effect of Quality Attribute Dimensionality c.
To observe the effect of quality attribute dimensionality, we varied the quality dimensionality c from 2 to 5. The results on IN and AC data are shown in Figure 9(c) and Figure 10(c) respectively. All algorithms are insensitive to the variation of attribute dimensionality. This is attributed to the fact that all

indexes used in those algorithms are spatial access methods. They only index spatial coordinates of all objects, and therefore are not affected much by the number of quality attributes in each object dataset. Here SJB remains to be the best solution.

Effect of Distance Constraint δ.
We also investigated into the effect of the distance constraint δ used in most endangered object queries. In this batch of experiments, we used the default settings and changed δ from 500 to 3000. Figure 11 reports the relevant results. On both attribute distributions, larger distance ranges result in marked performance degradation for IS and IS-Hil algorithms. This is because the ObjectScore algorithm (Algorithm 1) is called more often by these two algorithms when the distance range is larger. In contrast, almost no performance change is visible for the BFS, DFS and SJB algorithms. This implies that the variation of distance range does not affect the pruning power via the the R-trees exploited by those algorithms. Note that SJB still performs the best, indicating the strong pruning power of the upper bound score obtained from an entry's join list.

5 Conclusion and Future Work

In this paper we formalize a novel query, the most endangered object query (MEO), which takes into account both spatial distance constraint and multiple quality attributes. Given a competitor object set P and a candidate object set S, and a distance δ, the MEO query returns from S an object s such that it maximizes the number of objects of P dominating s within the δ-neighborhood (of s). It has important applications in business planning, online war games, wild animal protection, etc.

We propose several algorithms for processing MEO queries efficiently. The IS algorithm is an iterative search approach which requires that only P is indexed by an R-tree. To improve the performance, we index S by an aggregate R-tree, which enables effective pruning by using the aggregate count in each node entry. Then, best-first search (BFS) and depth-first search (DFS) for evaluating the query are studied. A spatial-join based algorithm (SJB) is also developed to process query fast. An extensive experiment study for the above methods is conducted on both synthetic and real datasets. Empirical results show that the SJB algorithm outperforms other solutions and it scales well for large datasets.

Several interesting directions exist for future research. First, we want to capture the realistic scenario that dominators (from P) tend to have more impact on a candidate object $s \in S$ when they are close to s. For this, the score function (in Definition 1) can be redefined by assigning higher weights to competitors that are close to s. The challenge is then to extend our proposed algorithms for such a weighted score function. Second, the set S of candidate locations can appear in other forms than a location set. For example, certain practical applications need to find the most endangered location(s) on a pre-defined trajectory of a wild animal or a military operation. Third, it is also of interest to define a generic query type that combines spatial locations and quality attributes. A

fundamental solution can be developed for such a generic definition, which can be instantiated to process concrete queries like MEO queries in this paper.

Acknowledgments We thank the anonymous reviewers and SSDBM 2009 PC Chair for their constructive comments.

References

1. S. Borzonyi, D. Kossmann, and K. Stocker. The skyline operator. In *Proc. ICDE*, pages 421–430, 2001.
2. T. Brinkhoff, H.-P. Kriegel, and B. Seeger. Efficient Processing of Spatial Joins Using R-Trees. In *Proc. SIGMOD*, pages 237–246, 1993.
3. Christian Böhm. A cost model for query processing in high dimensional data spaces. *ACM Trans. Database Syst.*, 25(2):129–178, 2000.
4. A. R. Butz. Alternative Algorithm for Hilbert's Space-Filling Curve. *IEEE Trans. Comput.*, C-20(4):424–426, 1971.
5. Y. Du, D. Zhang, and T. Xia. The optimal-location query. In *Proc. SSTD*, pages 163–180, 2005.
6. X. Huang and C. S. Jensen. In-route skyline querying for location-based services. In *Proc. W2GIS*, pages 120–135, 2004.
7. Z. Huang, H. Lu, B. C. Ooi, and A. K. H. Tung. Continuous skyline queries for moving objects. *TKDE*, 18(12):1645–1658, 2006.
8. N. Koudas and K. C. Sevcik. High Dimensional Similarity Joins: Algorithms and Performance Evaluation. In *Proc. ICDE*, pages 466–475, 1998.
9. C. Li, B. C. Ooi, A. K. H. Tung, and S. Wang. DADA: A data cube for dominant relationship analysis. In *Proc. SIGMOD*, pages 659–670, 2006.
10. C. Li, A. K. H. Tung, W. Jin, and M. Ester. On dominating your neighborhood profitably. In *Proc. VLDB*, pages 818–829, 2007.
11. B. Moon, H. V. Jagadish, C. Faloutsos, and J. H. Saltz. Analysis of the Clustering Properties of the Hilbert Space-Filling Curve. *TKDE*, 13(1):124–141, 2001.
12. D. Papadias, P. Kalnis, J. Zhang, and Y. Tao. Efficient OLAP Operations in Spatial Data Warehouses. In *Proc. SSTD*, pages 443–459, 2001.
13. D. Papadias, Y. Tao, G. Fu, and B. Seeger. An optimal and progressive algorithm for skyline queries. In *Proc. SIGMOD*, pages 467–478, 2003.
14. M. Sharifzadeh and C. Shahabi. The spatial skyline queries. In *Proc. VLDB*, pages 751–762, 2006.
15. T. Xia, D. Zhang, E. Kanoulas, and Y. Du. On computing top-t most influential spatial sites. In *Proc. VLDB*, pages 946–957, 2005.
16. M. L. Yiu, X. Dai, N. Mamoulis, and M. Vaitis. Top-k spatial preference queries. In *Proc. ICDE*, pages 1076–1085, 2007.
17. D. Zhang, Y. Du, T. Xia, and Y. Tao. Progressive computation of the min-dist optimal-location query. In *Proc. VLDB*, pages 643–654, 2006.
18. B. Zheng, K. C. K. Lee, and W.-C. Lee. Location-dependent skyline query. In *Proc. MDM*, pages 148–155, 2008.
19. M. Zhu, D. Papadias, J. Zhang, and D. L. Lee. Top-k spatial joins. *IEEE Trans. Knowl. Data Eng.*, 17(4):567–579, 2005.