Querying Spatial Data by Dominators in Neighborhood

Hua Lu[†] Man Lung Yiu[‡] Xike Xie[§]

[†]*Department of Computer Science, Aalborg University, Denmark*
[‡]*Department of Computing, Hong Kong Polytechnic University, Hong Kong*
[§]*School of Computer Science and Technology, University of Science and Technology of China*
[†]*luhua@cs.aau.dk,* [‡]*csmlyiu@comp.polyu.edu.hk,* [§]*xkxie@ustc.edu.cn*

Abstract

Spatial objects in reality are often associated with geographic locations (e.g., longitude and latitude) as well as multiple quality attributes. Quality attributes make it possible to compare spatial objects according to the dominance concept. 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 many contexts, an object's dominators in its neighborhood indicate the negative effect to the object. In this paper, we study the problem of querying spatial objects by their dominators in the neighborhood. We propose three meaningful score functions to quantify the negative effects of dominators in a spatial object's neighborhood. The most endangered object (MEO) query thus defined has multiple practical applications such as business planning, online war games, and wild animal protection. For processing MEO queries, we design several algorithms that require different indexes on spatial data sets. Each algorithm is generic and flexible such that each can support all three score functions (and even more) without significant changes. We conduct extensive experiments to evaluate the algorithms. The experimental results disclose the performance differences of the algorithms under various settings.

Keywords: Querying spatial data, Neighborhood dominators, Spatial data management

1. Introduction

Spatial objects in reality (e.g., hotels) are often associated with geographic locations (e.g., longitude and latitude) as well as multiple quality attributes (e.g., price and star). Conventionally, spatial objects are retrieved by spatial queries that select objects solely based on their locations and relevant distances. Typical spatial queries include range queries, nearest neighbor queries, spatial joins, closest pair queries, and so on. However, such queries fail to utilize the rich information captured by quality attributes.

As a matter of fact, quality attributes make it possible to compare spatial objects according to the dominance concept [2]. Specifically, an object p_1 dominates another object p_2 if p_1 is no worse than p_2 on all quality attributes and better than p_2 on at least one quality attribute. The dominance concept has been used to return the skyline [2] of a set of objects, which consists of all those objects that are not dominated by others.

In many contexts, a spatial object's dominators in its neighborhood indicate the threats that endanger the object or better choices that render the object disadvantaged and thus less attractive. For example, in online war games like World of Warcraft a troop is really in danger if it is positioned in a neighborhood where enemy troops are dominating in terms of multiple attributes like equipment and number of soldiers. As another example, a hotel is naturally less attractive to tourists if it is surrounded by dominating hotels (e.g., with lower prices and higher stars).

In this paper, we integrate the spatial aspect and the quality attributes of spatial objects and study a problem of finding objects with respect to the dominators in their neighborhood. We intend to find those spatial objects that are most affected (endangered or disfavored) with respect to their neighborhood dominators. Such objects should be given particular attention and treatment in order to survive or improve.

Generally speaking, our problem involves a set S of candidate objects and a set P of competitor objects. We intend to return from S those objects that are most affected by their nearby dominators from P.

We proceed to give a concrete motivation example. Suppose that a hotel chain is having financial crisis and needs to shut down one of its hotels in set S. Intuitively, a hotel is unlikely to make profit if its neighborhood contains a large number of dominating hotels. Therefore, such a hotel may be shut down. Note that dominators outside of the neighborhood do not have clear effect on a hotel's business. The locations of all hotels are shown in Figure 1, where s_i represents a hotel of the chain and h_i represents a competitor hotel in set P. In addition, each s_i's neighborhood is illustrated by a dashed circle centered at s_i with the radius of δ.

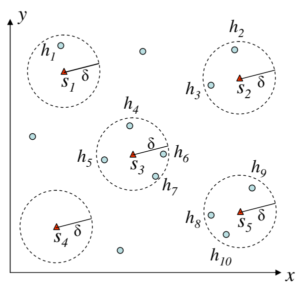


Figure 1: Spatial locations of hotels

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

Figure 2: Quality attributes of 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

Figure 3: Quality attributes of competitor set P

Figures 2 and 3 list the quality attributes (price and star) for hotels in candidate set S and hotels in competitor set P, respectively. Here, lower prices and higher stars are preferred. These preferences are used to determine dominators for each candidate hotel s_i in S. For example, candidate s_1 is dominated by competitors h_1, h_3, h_5 and h_9; s_3 is dominated by h_3, h_5 and h_9.

In order to compare candidate objects and select the most affected ones, we need to quantify the effect of the neighborhood dominators on each candidate object. The quantification is realized by score functions designed for candidate objects. A straightforward score function is to count the dominators and use the counts as scores for candidates. The intuition behind is that, the more dominators are there in the neighborhood, the more endangered is the object. Refer to Figure 1 again. Within the δ-neighborhood, candidate hotel s_1 is only dominated by h_1; s_3 is dominated by the nearby h_5 only; s_5 is dominated by both h_8 and h_{10}. In contrast, s_2 has no neighborhood dominators and s_4 has no neighbors. As a result, hotel s_5 will be returned according to the counting based score function.

Furthermore, we define two other score functions for candidate objects. The *distance sensitive score* gives more weights to dominators closer to an object when quantifying the effect for the object. The *disadvantage aware score* quantifies the quality difference between an object and its neighborhood dominators.

We also propose algorithms to find those most affected candidate objects from S. The algorithms are generic and flexible such that each of them can accommodate all the three aforementioned score functions. They can actually deal with more score functions as long as a function is with necessary properties.

In addition to the business planning example, querying spatial objects with respect to neighborhood dominators also offers useful results for other fields. For online war games (e.g., World of Warcraft), P is the set of enemy troops and S is the set of ally troops. Each troop can be described by multiple quality attributes like solider number, equipment level, etc. Accordingly, we can identify the most endangered troop of S that needs 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. Accordingly, we can identify the animals in the greatest need of protection.

We make the following contributions in this paper.

- We define three meaningful score functions for spatial objects to quantify the effects of their neighborhood dominators.

- We formalize the generic problem of querying spatial objects according to the score functions with respect to neighborhood dominators.

- We derive effective search bounds for the score functions such that the functions can be supported by generic algorithm frameworks for querying spatial objects.

- We discuss how to support the generic distance and other score functions in our frameworks for querying of spatial objects.

- We conduct extensive experiments to evaluate our proposals. The experimental results disclose the most scalable and efficient algorithm framework for all score functions.

This paper extends our previous work [18] with substantial new contributions. First, this paper improves previous algorithms with more efficient pruning (Section 3.5). Second, this paper defines a distance sensitive score function to measure the neighborhood threats and presents search bounds that can be used by generic algorithm frameworks (Section 4). Third, this paper also defines a score function and corresponding search bounds that consider an object's disadvantage compared to its neighborhood threats (Section 5). Fourth, this paper discusses how to support the generic distance and what properties score functions should have in the proposed frameworks (Section 6). Last, extensive experiments are conducted to evaluated the new proposals (Section 7).

The remainder of this paper are organized as follows. Section 2 gives preliminaries including the problem formulation and an overview of the algorithm frameworks. Section 3 details the counting based score function for neighborhood threats and the search bounds. Section 4 details the distance sensitive score function and the search bounds. Section 5 details the disadvantage aware score function and the search bounds. Section 6 discusses the generic distance and score function properties. Section 7 reports on extensive experiments of our proposals on both synthetic and real data. Section 8 reviews the related work. Finally, Section 9 concludes the paper.

2. Preliminaries

This section formulates the problems (Section 2.1) and gives an overview on the score functions and query algorithms (Section 2.2).

2.1. Problem Formulation

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 and $\psi[i]$ 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 [2], a quality vector ψ *dominates* another one ψ' (denoted as $\psi \prec \psi'$) iff:

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

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 $o.\psi \prec o'.\psi$. We use $\mathbb{O}$ to denote the set of all spatial objects in a space of interest.

Definition 1. ***(Neighborhood Dominators)*** *Given a spatial object set P, a spatial object s, and a distance threshold δ, the neighborhood dominators are those P objects that dominate s and lie in its δ-neighborhood:*

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

Definition 2. ***(Generic Neighborhood Threat Score)*** *Given a spatial object set P, a spatial object s, and a distance threshold δ, the generic neighborhood threat score of s with respect to P and δ is defined as:*

$$\Phi_{P,\delta}(s) = f(\Delta_{P,\delta}(s), s)$$

Here, f is a generic score function on a spatial object and all its neighboring dominators, defined as follows:

$$f : 2^{\mathbb{O}} \times \mathbb{O} \rightarrow \mathbb{R} \tag{2}$$

When the context is clear, we and use $\Phi(s)$ to denote object s's neighborhood threat score. The concrete values of $\Phi(s)$ are determined by the concrete function that is used for f.

In this paper, we present three concrete instances for the generic score function f. The first score function equally counts all the dominators in an object s's neighborhood (Section 3). It applies to the scenarios where only the number of dominators is of importance. The second score function takes into account the distance decay effect when counting the dominators in object s's neighborhood (Section 4). It applies to the scenarios where nearby dominators mean more threats and faraway ones less. Instead of counting dominators, the third score function measures object s's quality disadvantage with respect to the dominators in its neighborhood (Section 5). It applies to the scenarios where the quality difference matters between object s and its dominators.

Next, we define the *most endangered object query* (MEO) by using the generic score function.

Definition 3. ***(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., $\forall\ s' \in S$, $\Phi_{P,\delta}(s) \geq \Phi_{P,\delta}(s')$.*

The MEO query finds the spatial object with the highest neighborhood threat score. In this paper, we design efficient algorithms for the MEO query. We proceed to give an overview of the proposed algorithms.

2.2. Overview of Score Functions and Algorithmic Solutions

Table 1 lists the notations used throughout the paper.

Notation	**Meaning**
P	the set of competitor objects
S	the set of candidate objects
$\psi \prec \psi'$	a quality vector ψ dominates another one ψ'
$dist(o, o')$	Euclidean distance between objects o and o'
$mindist(e, e')$	minimum distance between R-tree entries e and e'
$\Delta_{P,\delta}(s)$	s's δ-neighborhood dominators
$\Phi_{P,\delta}(s)$	δ-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: Notations

Our first algorithm is an iterative search algorithm (IS for short). Specifically, it loops on the candidate set S. For each candidate object s_i encountered, it searches the competitor set P to get all the dominators in s_i's neighborhood and to calculate the score. When the loop on S is finished, the candidate object with the highest score is returned. The IS algorithm requires an R-tree index on P but no index on S.

The second is a best-first search algorithm (BFS for short). It requires two R-trees: R_S on S and R_P on P. It searches the R-tree R_S on S in a best-first fashion with a priority queue controlling the node access and expansion. The key used in the queue is the score for a tree entry e_S, which is an upper bound score for all candidate objects in e_S. The BFS algorithm stops when a leaf entry is popped up from the priority queue since it means the object with the highest score is found.

The third is a spatial join based algorithm (SJB for short). Requiring R-trees on both object sets, it processes the object query in a spatial join manner. In particular, an entry e_S from R_S is paired with all R_P entries that may have neighborhood dominators for an object in e_S. Entry e_S is associated with a score that is the upper bound for all objects in it. The join is pushed downwards along the two trees, with priority given to R_S entries with higher scores. The SJB algorithm stops when a leaf entry from R_S is encountered, and the corresponding object is returned.

Score Function	**IS Approach**	**BFS Approach**	**SJB Approach**
Dominator Counting (Def. 4)	Alg. 2 + Alg. 1 R-tree for P only, S not indexed	Alg. 4 + Alg. 3/Alg. 6 + Alg. 1 aggregate R-tree for P, R-tree for S	Alg. 5 + Alg. 1 aggregate R-tree for P, R-tree for S
Distance Sensitive (Def. 5)	Alg. 2 + Alg. 7 R-tree for P only, S not indexed	Alg. 4 + Alg. 8 + Alg. 7 aggregate R-tree for P, R-tree for S	Alg. 5 + Alg. 7 aggregate R-tree for P, R-tree for S
Disadvantage Aware (Def. 10)	Alg. 2 + Alg. 9 R-tree for P only, S not indexed	Alg. 4 + Alg. 10 + Alg. 9 min-aggregate R-tree for P, max-aggregate R-tree for S	Alg. 5 + Alg. 9 min-aggregate R-tree for P, max-aggregate R-tree for S

Table 2: Score functions and algorithms

Each of these three algorithms is able to accommodate the different s-core functions defined in this paper. When one score function is changed to another, only slight adjustment is needed for each algorithm. Nevertheless, BFS and SJB use different methods to estimate upper bound scores for all objects in an R-tree entry. The following sections detail the definition and the algorithms for each score function. A summary of all score functions and algorithms are given in Table 2.

3. Querying by Neighborhood Dominator Count

In this section, we only consider the count of dominators in a neighborhood as the threat to an object. Section 3.1 defines the dominator count score function. Section 3.2 details the iterative algorithm, Section 3.3 presents the best-first algorithm, Section 3.4 elaborates on the spatial join algorithm, and Section 3.5 discusses improvements to the best-first algorithm.

3.1. Dominator Count Score Definition

In many application scenarios, it makes sense to count the dominators within an object s's neighborhood and use that count as the indication of how s is threatened. For example, in wild animal protection, the more dominators (animals with multiple better abilities like speed, weight, age, etc.) in an animal's neighborhood, the more endangered it is. It is therefore important to identify those animals that have the highest number of neighborhood dominators and save them accordingly. Motivated as such, we give the following definition.

Definition 4. ***(Neighborhood Dominator Count Score)*** *Given a spatial object set P, a spatial object s, and a distance threshold δ, the neighbor-*

hood dominator count score of s *with respect to* P *and* δ *is defined as:*

$$\Phi^{DC}_{P,\delta}(s) = |\Delta_{P,\delta}(s)| \tag{3}$$

Whenever the context becomes clear, we drop the superscripts/subscripts and use $\Phi(s)$ to denote the neighborhood dominator count score for s.

3.2. Iterative Search Algorithm

In this section, we assume that the data set P is indexed by an R-tree R_P and the data set 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.

ObjectScoreDC (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), the algorithm checks 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), the algorithm reads the child node corresponding to e_P, and recursively processes each of its entry e'_P if the minimum distance $mindist(e'_P, s)$ from e'_P to s is within δ.

Algorithm 1 ObjectScoreDC(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** $dist(s, e_P) \leq \delta$ and $e_P.\psi \prec s.\psi$ **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** $mindist(s, e'_P) \leq \delta$ **then**
9:             $v := v+$ObjectScoreDC$(s, e'_P, \delta)$
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 the competitor set P, (ii) the candidate object set S, and (iii) the distance

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

```
1: $meo$ :=null; $\gamma := 0$
2: for each object $s \in S$ do
3:     $\Phi(s)$ :=ObjectScoreDC($s, R_P.root, \delta$)
4:     if $\Phi(s) > \gamma$ then
5:         $\gamma := \Phi(s)$
6:         $meo := s$
7: return $meo$
```

δ. The object *meo* is used to keep track of the result object found so far, and the value γ corresponds to the score of *meo*. At line 1, *meo* is initialized to null and γ to 0. 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, 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, 20], and then processes them by IS.

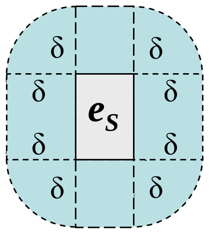
3.3. Best-First Search Algorithm

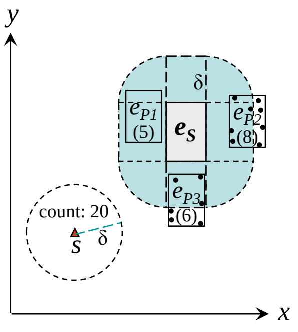
Observe that IS algorithm processes every object once in the set S. We now propose to index the set S by an R-tree R_S and develop an efficient method to prune unqualified subtrees of R_S that cannot contribute to the result.

In order to support efficient counting operations, we index the data set P by an aggregate `COUNT` R-tree R_P [21]. Specifically, each non-leaf entry e_P in 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 4 shows the spatial extent of e_S as a rectangular region. We intend 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.

First, we introduce the concept of δ-Minkowski region [1] of e_S, denoted by $\Xi(e_S, \delta)$, which is the set of possible locations whose minimum distance





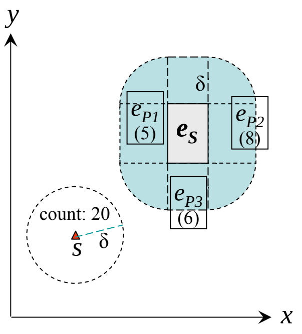


Figure 4: $\Xi(e_S, \delta)$ Figure 5: Pruning Rule 1 Figure 6: Pruning Rule 2

from e_S is within the distance δ.

$$\Xi(e_S, \delta) = \{t \in \mathbb{R}^2 \mid mindist(t, e_S) \leq \delta\} \tag{4}$$

The region $\Xi(e_S, \delta)$ is illustrated in Figure 4 as the region extended from the rectangle e_S by the distance δ. Given the data set 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 $\Xi(e_S, \delta)$.

$$\Phi^+_{P,\delta}(e_S) = |\{o \in P \mid o \subseteq \Xi(e_S, \delta)\}| \tag{5}$$

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 data set 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 1. *Let s be an object that falls into e_S. According to Definitions 1 and 4, 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 an 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 **EntryScoreDC** 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 two purposes, depending on whether e_S is a leaf entry or not. If e_S is a leaf entry, 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, and lines 4–11 are used to compute the upper bound score $\Phi^+(e_S)$ of e_S. The algorithm checks whether the region $\Xi(e_S, \delta)$ contains e_P. If so, each object in the subtree of e_P is guaranteed to fall in $\Xi(e_S, \delta)$. Thus, the counter v is incremented by the count $e_P.count$, without visiting the subtree of e_P. If not, the algorithm needs 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 EntryScoreDC(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:     $v :=$ObjectScoreDC$(e_S, e_P, \delta)$
4: else                                              ▷ $e_S$ is a non-leaf entry
5:     if $\Xi(e_S, \delta)$ contains $e_P$ then
6:         $v := e_P.count$
7:     else
8:         if $e_P$ is a non-leaf entry then
9:             read the child node $CN$ pointed to by $e_P$
10:            for each child $e'_P$ in $CN$ do
11:                if $\Xi(e_S, \delta)$ intersects $e'_P$ then
12:                    $v := v+$EntryScoreDC$(e_S, e'_P, \delta)$
13: return $v$
```

Figure 5 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 EntryScoreDC algorithm. Here, the aggregate R-tree R_P (of the data set P) only has the non-leaf entries e_{P1}, e_{P2}, e_{P3}, whose associated count values are 5, 8 and 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 EntryScoreDC algorithm for computing it. We continue to present a pruning rule for reducing the search space, and then develop two algorithms for solving MEO query 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 most endangered 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 5 to illustrate this pruning rule. Suppose that we have already examined object s and computed its exact value $\Phi(s) = 20$ (by the EntryScoreDC 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 EntryScoreDC 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 *best-first search* (BFS). The pseudo code of BFS is shown in Algorithm 4. It employs a max-heap H 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 EntryScoreDC 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 EntryScoreDC($R_P.root, e_S, \delta$) (stated in Lemma 1).

3.4. Spatial Join Based Algorithm

As in Section 3.3, 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 the BFS algorithm needs 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

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($H, \langle R_S.root, |P| \rangle$)
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($H, \langle e'_S, \text{EntryScoreDC}(e'_S, R_P.root, \delta) \rangle$)
9:     else
10:        return $e_S$
```

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:

$$\Phi^*_{P,\delta}(e_S) = \sum_{e' \in e_S.JL} e'.count \tag{6}$$

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 2. *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 $\Phi^*_{P,\delta}(e_S) \geq \Phi_{P,\delta}(s)$.*

Based on this lemma we have the following pruning rule.

Pruning Rule 2. *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 $\Phi_{P,\delta}(s) > \Phi^*_{P,\delta}(e_S)$, then the entry e_S can be safely pruned.*

We give an example on exploiting the pruning rule. Figure 6 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 6.

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 neighborhood 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($H, \langle e_{root}, e_{root}.JL, 0\rangle$)
4: while $H$ is not empty do
5:     $\langle e_S, e_S.JL\rangle :=$ 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:                $v := v+$ObjectScoreDC($e_S, e_j, \delta$)
13:            enheap($H, \langle e_S, null, v\rangle$)
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:            $v := 0$; $e_i.JL := \emptyset$
18:            for each $e_j$ in $e_S.JL$ do
19:                if $\Xi(e_i, \delta)$ contains $e_j$ then
20:                    add $e_j$ to $e_i.JL$;
21:                    $v := v + e_j.count$
22:                else
23:                    read the child node $CN_P$ pointed to by $e_j$
24:                    for each child $e'$ in $CN_P$ do
25:                        if $\Xi(e_i, \delta)$ intersects $e'$ then
26:                            add $e'$ to $e_i.JL$;
27:                            $v := v + e'.count$
28:            enheap($H, \langle e_i, e_i.JL, v\rangle$)
```

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 continues, the R_S entries are enheaped with join lists of smaller coverage, thus giving tighter upper bounds of neighborhood dominator counts which favors pruning.

3.5. Optimizations for BFS: Efficient Upper Bound Computation

In Section 3.3, we propose algorithms that search the R-tree R_S of objects set S. Specifically, BFS algorithm calls EntryScoreDC(.) (Algorithm 3) to estimate the upper bound score for a tree entry e_S in R_S. Given an entry e_S, EntryScoreDC(.) counts the exact number of objects in P that lie in e_S's δ-Minkowski region $\Xi(e, \delta)$. For an R-tree entry e_P from R_P that intersects $\Xi(e, \delta)$, EntryScoreDC(.) recurs to access e_P's child (descent) entries until the R_P entry being visited is inside $\Xi(e, \delta)$.

As a matter of fact, this exact counting is not always necessary. Specifically, if we do not go deeper to its children from an entry e_P that intersects $\Xi(e, \delta)$, we will overestimate the upper bound score but save R-tree traversal cost. This overestimation will not cause any false negatives, i.e., it will not miss any candidates, although it may weaken the pruning power of Pruning Rule 1. The reasoning behind the correctness is similar to that behind the use of the join list in the join based algorithm in Section 3.4.

Motivated by the aforementioned observations, we improve the BFS algorithm by modifying the upper bound cost estimation in EntryScoreDC(.). The improved version, formulated in Algorithm 6, simplifies the visit to a non-leaf node pointed to by e_P by adding to the score the object count in e_P's each child node (line 12). In this way, it avoids recursively processing each child node under e_P. In other words, the algorithm stops at each R_P entry e'_P that intersects R_S entry e_S, which results in the similar effect as if the join list is maintained for e_S. Therefore, the upper bound score is overestimated as $\Phi^*_{P,\delta}(e_S)$ (Equation 6).

Pruning Rule 2 guarantees the correctness of the BFS algorithm that calls Algorithm 6 to get the overestimated score. The improvement presented in this section renders the BFS algorithm to behave more like the SJB algorithm. In particular, less computation will be spent in counting the dominators precisely for an R-tree entry e_S in the former two algorithms. We experimentally evaluate the effectiveness of the improvement in Section 7.

4. Querying by Distance Sensitive Score

In this section, we take into account the distance decay effect between candidate objects and the threats nearby. We define the distance sensitive neighborhood threat score in Section 4.1. From Section 4.2 to Section 4.4, we discuss how the query processing techniques presented in Section 3 can be adapted to support the MEO query using the distance sensitive score.

Algorithm 6 EntryScoreDC2(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:     $v :=$ObjectScoreDC$(e_S, e_P, \delta)$
4: else                                              ▷ $e_S$ is a non-leaf entry
5:     if $\Xi(e_S, \delta)$ contains $e_P$ then
6:         $v := e_P.count$
7:     else
8:         if $e_P$ is a non-leaf entry then
9:             read the child node $CN$ pointed to by $e_P$
10:            for each child $e'_P$ in $CN$ do
11:                if $\Xi(e_S, \delta)$ intersects $e'_P$ then
12:                    $v := v + e'_P.count$
13: return $v$
```

4.1. Distance Sensitive Score Definition

Given a location q, the aforementioned neighborhood threat score (Definition 4 in Section 3) simply counts the dominators (i.e., threats) in q's neighborhood but does not consider their distances from q. In many scenarios the distance between a threat and an object matters or even is critical. An example is shown in Figure 7, where s_1 and s_2 are two soldiers while dominating enemies are captured as dots. We cannot differentiate s_1 and s_2 using the counting based score since both of them have five neighborhood threats. However, it is apparent that s_1 is more endangered because s_1's enemies are much closer compared to those to s_2. In such cases, given a particular object s, the nearby threats are more dangerous than faraway threats.

Motivated as such, it makes sense to differentiate the threats that are at different distances from an object s in consideration. We define the distance sensitive neighborhood threat score as follows.[1]

Definition 5. ***(Distance Sensitive Neighborhood Threat Score)*** *Given a spatial object set P, a spatial object s, and a distance threshold δ, the distance sensitive neighborhood threat score of s with respect to P and δ is*

[1]Our distance sensitive score partially resembles the influence score defined in [36]. In particular, ours uses a distance threshold to restrict the search to an object's neighborhood, whereas the influence score based search [36] expands to the entire space.

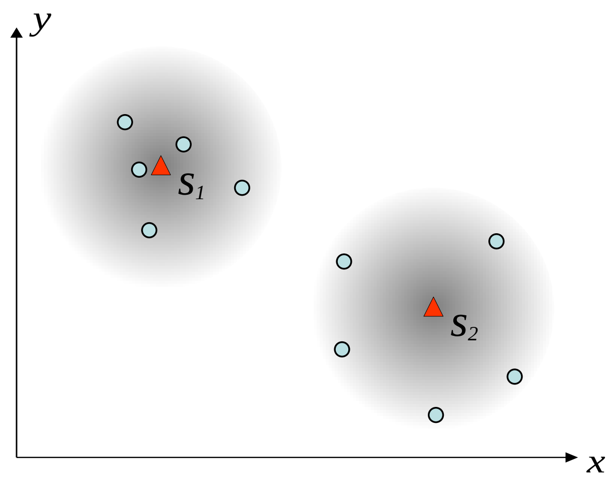


Figure 7: Example of Distance Sensitive Score

defined as:

$$\Phi^{DS}_{P,\delta}(s) = \sum_{o \in \Delta_{P,\delta}(s)} 2^{-dist(s,o)} \tag{7}$$

In the following sections, we extend the MEO query processing techniques proposed in Section 3 to process MEO query that uses the distance sensitive score. Again, we drop the superscripts/subscripts and use $\Phi(s)$ to denote the score for s when the context is clear.

4.2. Query Processing by Iterative Search

The iterative search (IS in Algorithm 2) still works for the MEO query that adopts the distance sensitive score definition. The only change we need to do is to the ObjectScoreDC function (Algorithm 1). In particular, we use Definition 5 instead of Definition 4 on line 4. We name the modified algorithm ObjectScoreDS (Algorithm 7).

4.3. Query Processing by Best-First Search

In order to use the best-first search to process the MEO query with distance sensitive scores, we still use an aggregate `COUNT` R-tree R_P to index data set P and an R-tree R_S to index data set S. We also need to derive an appropriate upper bound score for the best-first search to work with object groups organized as R-tree nodes. Therefore, we introduce the concept of Minkowski entry set as follows.

Algorithm 7 ObjectScoreDS(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** $dist(e_P, s) \leq \delta$ and $e_P.\psi \prec s.\psi$ **then**
4:         $v := 2^{-dist(e_P,s)}$
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** $mindist(e'_P, s) \leq \delta$ **then**
9:             $v := v$+ObjectScoreDS$(s, e'_P, \delta)$
10: **return** $v$
```

Definition 6. ***(Minkowski Entry Set)*** *Given a distance threshold δ, an aggregate R-tree R_P for data set P, and an entry e_S from R-tree R_S for data set S, e_S's Minkowski entry set $\Omega_{P,\delta}(e_S)$ consists of R_P entries that satisfy all the following conditions:*

1. *$\forall e_P \in \Omega_{P,\delta}(e_S), e_P \subseteq \Xi(e_S, \delta)$, i.e., e_P is covered by e_S's δ-Minkowski region;*
2. *$\forall e_P, e'_P \in \Omega_{P,\delta}(e_S)$, if $e_P \neq e'_P$ then they are not on the same path in R_P;*
3. *$\forall e'_P \in R_P$, if $e'_P \subseteq \Xi(e_S, \delta)$, then either $e'_P \in \Omega_{P,\delta}(e_S)$ or $e'_P \notin \Omega_{P,\delta}(e_S)$ but $\exists e_P \in \Omega_{P,\delta}(e_S)$ such that e_P is e'_P's ancestor in R_P.*

Specifically, the first condition above requires that any entry in e_S's Minkowski entry set must spatially be in e_S's δ-Minkowski region. This ensures that all threats in the δ-neighborhood of any object in e_S will be included when e_S is processed. The second condition above specifies that different entries in e_S's Minkowski entry set cannot come from the same path in R-tree R_P. The second and third conditions together imply that only the highest ancestor on a path in R_P can enter e_S's Minkowski entry set. As a result, the Minkowski entry set contains all the highest level R_P entries that are covered by e_S's δ-Minkowski region.

With the concept of Minkowski entry set, we define the upper bound distance sensitive score for an R_S entry e_S for the candidate set S as follows.

Definition 7. ***(Upper Bound Distance Sensitive Score)***

$$\Phi^{+}_{P,\delta}(e_S) = \sum_{e_P \in \Omega_{P,\delta}(e_S)} e_P.count \cdot 2^{-mindist(e_S,e_P)}$$

The correctness of this upper bound cost is guaranteed by the following lemma.

Lemma 3. *$\Phi^{+}_{P,\delta}(e_S) \geq \Phi^{DS}_{P,\delta}(s)$ holds for any object s that falls into e_S.*

Proof 3. *$\forall o \in \Delta_{P,\delta}(s)$, it holds that $o \in \Xi(e_S, \delta)$. According to Definition 6, o's all neighborhood dominators in $\Delta_{P,\delta}(s)$ correspond to m R-tree entries $e_{P1}, e_{P1}, \ldots, e_{Pm}$ $(1 \leq m \leq |\Delta_{P,\delta}(s)|)$ in $\Omega_{P,\delta}(e_S)$. We use $\Omega'_{P,\delta}(e_S)$ to denote $\{e_{P1}, e_{P1}, \ldots, e_{Pm}\}$.*

According to Definition 7, we have $\Phi^{+}_{P,\delta}(e_S) = \sum_{e_P \in \Omega_{P,\delta}(e_S) \setminus \Omega'_{P,\delta}(e_S)} e_P.count \cdot 2^{-mindist(e_S,e_P)} + \sum_{e_P \in \Omega'_{P,\delta}(e_S)} e_P.count \cdot 2^{-mindist(e_S,e_P)} \geq \sum_{e_P \in \Omega'_{P,\delta}(e_S)} e_P.count \cdot 2^{-mindist(e_S,e_P)} = \sum_{i=1}^{m} e_{Pi}.count \cdot 2^{-mindist(e_S,e_{Pi})}$.

As $mindist(s, e_P) \geq mindist(e_S, e_P)$ for any object s in e_S, it holds $\Phi^{+}_{P,\delta}(e_S) \geq \sum_{i=1}^{m} e_{Pi}.count \cdot 2^{-mindist(s,e_{Pi})}$.

Further, $\forall e_{Pi} \in \Omega'_{P,\delta}(e_S)$, suppose e_{Pi} contains n $(1 \leq n \leq |\Delta_{P,\delta}(s)|)$ dominators $o_1, o_2, \ldots, o_n$ from $\Delta_{P,\delta}(s)$. As $e_{Pi}.count \geq n$ and $mindist(s, e_{Pi}) \leq mindist(s, o_j)$ for $1 \leq j \leq n$, we have $e_{Pi}.count \cdot 2^{-mindist(s,e_{Pi})} \geq \sum_{j=1}^{n} \cdot 2^{-mindist(s,o_j)}$.

To put everything together, it holds $\Phi^{+}_{P,\delta}(e_S) \geq \sum_{j=1}^{|\Delta_{P,\delta}(s)|} \cdot 2^{-mindist(s,o_j)} = \Phi_{P,\delta}(s)$. The lemma is proved.

With the new upper bound cost, we can still use the best-first search (Algorithm 4) for the MEO query with the new score definition. However, the best-first search algorithm should call the EntryScoreDS2 function (Algorithm 8) instead of the EntryScoreDC or EntryScoreDC2 functions on its line 8. The EntryScoreDS2 function makes use of the new upper bound cost derived in this section.

4.4. Query Processing by Spatial Join

The MEO query with the distance sensitive score can also be processed by the spatial join based method. The key point here is to accordingly derive a new upper bound score to be used by spatial join. Given R-tree R_S for the set of candidate objects S and aggregate R-tree R_P for the set of competitor objects P, we define the *join list* for an entry e_S from R_S in the same way as described in Section 3.4. The upper bound score is defined as follows.

Algorithm 8 EntryScoreDS2(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:      $v :=$ObjectScoreDS($e_S, e_P, \delta$)
4:  else                                              ▷ $e_S$ is a non-leaf entry
5:      if $\Xi(e_S, \delta)$ contains $e_P$ then
6:          $v := e_P.count \cdot 2^{-mindist(e_S, e_P)}$
7:      else
8:          if $e_P$ is a non-leaf entry then
9:              read the child node $CN$ pointed to by $e_P$
10:             for each child $e'_P$ in $CN$ do
11:                 if $\Xi(e_S, \delta)$ intersects $e'_P$ then
12:                     $v := v + e'_P.count \cdot 2^{-mindist(e_S, e'_P)}$
13: return $v$
```

Definition 8. ***(Upper Bound Distance Sensitive Score for Spatial Join)***

$$\Phi^*_{P,\delta}(e_S) = \sum_{e_P \in e_S.JL} e_P.count \cdot 2^{-mindist(e_S, e_P)} \tag{8}$$

The correctness of this upper bound cost is guaranteed by the following lemma.

Lemma 4. *$\Phi^*_{P,\delta}(e_S) \geq \Phi_{P,\delta}(s)$ holds for any object s that falls into e_S.*

Proof 4. *Note that e_S's join list $e_S.JL$ contains all dominators that are in the neighborhood of s, i.e., $\Delta_{P,\delta}(s)$ are covered by all entries in $e_S.JL$. Therefore, this lemma can be proved by the same reasoning as the proof for Lemma 3.*

This upper bound cost allows us to adapt the spatial join (Algorithm 5) to process the MEO query with distance sensitive scores. In particular, we use the count aggregate R-tree to index P instead, replace the function ObjectScore(.) in line 12 with ObjectScoreDS(.) (Algorithm 7), $v := v + e_j.count$ in line 21 with $v := v + e_j.count \cdot 2^{-mindist(e_i, e_j)}$, and $v := v + e'.count$ in line 27 with $v := v + e'.count \cdot 2^{-mindist(e_i, e')}$.

5. Querying by Disadvantage Aware Score

In this section, we consider an object's disadvantage with respect to its neighborhood threats. We quantify the disadvantage and define the disadvantage aware score for objects in Section 5.1. From Sections 5.2 to 5.4, we propose query processing techniques for the MEO query based on the disadvantage aware score.

5.1. Disadvantage Aware Score Definition

An object is dominated by others due to its disadvantage on the quality atrributes. In many cases, it is interesting to know how disadvantaged an object s is compared to its dominators (threats). For example, in the context of digital war systems, it is critical to find the most disadvantaged battle unit with respect to its nearby enemies. Such disadvantages lie in the number of soldiers, the amount of firepower, the level of equipment, etc. In such cases, we need to quantify the disadvantages in order to identify the most disadvantaged objects and to take necessary actions. A meaningful and simple way to do it is formalized as follows.

Definition 9. ***(Disadvantage)*** *Given two c-dimensional quality vectors ψ and ψ', if $\psi \prec \psi'$, we say ψ''s disadvantage with respect to ψ is $\tau(\psi', \psi) = \sum_{1 \leq i \leq c}(\psi'[i] - \psi[i])$;*[2] *otherwise, we stipulate $\tau(\psi', \psi) = 0$.*

The definition above measures the Manhattan distance between two quality vectors ψ and ψ'.[3] The definition allows us to define the disadvantage aware neighborhood threat score as follows.

Definition 10. ***(Disadvantage Aware Neighborhood Threat Score)*** *Given a spatial object set P, a spatial object s, and a distance threshold δ, the disadvantage aware neighborhood threat score of s with respect to P and δ is defined as:*

$$\Phi^{DA}_{P,\delta}(s) = \max_{p \in \Delta_{P,\delta}(s)} \sum_{1 \leq i \leq c} (s.\psi[i] - p.\psi[i]). \tag{9}$$

[2] We assume that the domain on each of the c quality dimensions is normalized to the unit range $[0, 1]$ such that the summation makes sense.

[3] This way follows a previous work [17]. Nevertheless, the techniques proposed in this paper can be adapted to other monotonic disadvantage measurements.

Accordingly, we can issue the MEO query using the disadvantage aware score. In the following sections, we give query processing algorithms for such MEO queries. Again, we drop the superscripts/subscripts and use $\Phi(s)$ to denote the score for s when the context is clear.

5.2. Query Processing by Iterative Search

We first consider the iterative search for the MEO query using the disadvantage aware score. To this end, the IS algorithm (Algorithm 2 in Section 3) still can be used. However, we need to change the algorithm ObjectScore for computing object scores. To compute the disadvantage aware object score, ObjectScore (line 3 in Algorithm 2) should be replaced by ObjectScoreDA (Algorithm 9).

Algorithm 9 ObjectScoreDA(Object s, Entry e_P of P's R-tree R_P, Distance δ)

```
1: $v := 0$
2: if $e_P$ is a leaf entry then
3:     if $dist(s, e_P) \le \delta$ and $e_P.\psi \prec s.\psi$ then
4:         $v := \sum_{1 \le i \le c} s.\psi[i] - e_p.\psi[i]$
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 $mindist(s, e'_P) \le \delta$ then
9:             $v := \max(v, \text{ObjectScoreDA}(s, e'_P, \delta))$
10: return $v$
```

5.3. Query Processing by Best-First Search

In this section, we consider the best-first search for the MEO query using the disadvantage aware score. To this end, we need to augment the R-tree for candidate set S and that for competitor set P. Specifically, P's R-tree R_P is a min-aggregate R-tree that still groups the nodes according to their spatial extent but augments each node entry e with an additional c-dimensional vector $e.min$ whose i-th value is recursively defined as:

$$e.min[i] = \min\{e'.min[i] \mid e' \in e\text{'s child nodes}\} \tag{10}$$

The value $e.min[i]$ indicates the minimum i-th quality attribute value of all objects in e's subtree.

Compared to P's R-tree R_P, S's R-tree R_S is almost the same except that each of its entry e is augmented with a maximum vector $e.max$ whose i-th value is recursively defined as:

$$e.max[i] = \max\{e'.min[i] \mid e' \in e\text{'s child nodes}\} \tag{11}$$

The best-first search framework (Algorithm 4) can be reused but the entry score should be computed in a way with respect to Definition 10. It is formalized as EntryScoreDA in Algorithm 10. On line 12 of Algorithm 10, an upper bound disadvantage aware score is used for fast pruning an unqualified tree entry e'_P in the min-aggregate R-tree R_P. That upper bound score is defined in Definition 11.

Algorithm 10 EntryScoreDA(Entry e_S of the max-aggregate R-tree R_S on S, Entry e_P in the min-aggregate R-tree R_P on P, Distance δ)

```
1: $v := 0$
2: if $e_S$ is a leaf entry then
3:     $v :=$ObjectScoreDA$(e_S, e_P, \delta)$
4: else                                              ▷ $e_S$ is a non-leaf entry
5:     if $\Xi(e_S, \delta)$ contains $e_P$ then
6:         $v :=$ UBScore$(e_S, e_P)$
7:     else
8:         if $e_P$ is a non-leaf entry then
9:             read the child node $CN$ pointed to by $e_P$
10:            for each child $e'_P$ in $CN$ do
11:                if $\Xi(e_S, \delta)$ intersects $e'_P$ then
12:                    if UBScore$(e_S, e'_P) > v$ then
13:                        $v := \max(v, \text{EntryScoreDA}(e_S, e'_P, \delta))$
14: return $v$
```

Definition 11. ***(Upper Bound Disadvantage Aware Score)*** *Given an entry e_S of the max-aggregate R-tree R_S on spatial objects S and an entry e_P of the min-aggregate R-tree R_P on competitors P, the upper bound disadvantage aware score for any object $s \in e_S$ and any dominator $p \in e_P$ is*

$$\text{UBScore}(e_S, e_P) = \sum_{1 \le i \le c} \phi(e_S.max[i], e_P.min[i]),$$

where

$$\phi(v, w) = \begin{cases} v - w, \text{if } v > w; \\ 0, \text{otherwise}. \end{cases} \quad (12)$$

The correctness of this upper bound score is guaranteed by the following lemma.

Lemma 5. UBScore$(e_S, e_P) \geq \tau(s.\psi, p.\psi)$, $\forall s \in e_S$ *and* $\forall p \in e_P$.

It is straightforward to prove the lemma. An example is given in Figure 8, where smaller values are preferred in both quality dimensions QD_1 and QD_2. Consider an arbitrary point s in e_S and an arbitrary point p in e_P. The Manhattan distance between s and p is always upper bounded by that between the two corners $e_S.max$ and $e_P.min$.

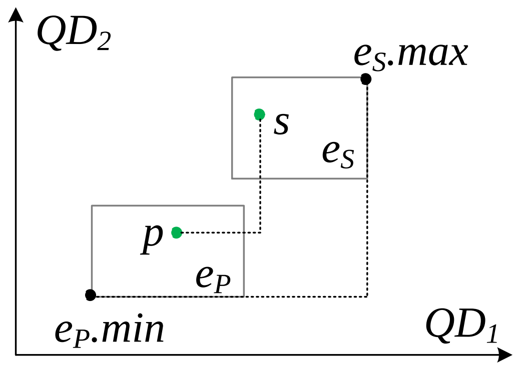


Figure 8: Entry e_S's Upper Bound Disadvantage with respect to Entry e_P

5.4. Query Processing by Spatial Join

In this section, we consider the spatial join for the MEO query using the disadvantage aware score. As the object score definition is substantially different from the previous counterparts, the spatial join framework needs to take that into account and to incorporate necessary modifications. To this end, we keep the overall procedure of Algorithm 5 but modify a few particular places of it. First, we use the max-aggregate R-tree and min-aggregate R-tree for S and P, respectively, in the input. Second, line 12 is changed to $v := \max(v, \text{ObjectScoreDA}(e_S, e_j, \delta))$ in order to compute the object score when an object (represented by entry e_S) is encountered. Third, line 21 is changed to $v := \max(v, \text{UBScore}(e_S, e_j))$ to overestimate the score for all objects in e_S by considering the largest possible disadvantage of e_S with respect to an entry e_P from P's R-tree R_P. Likewise, line 27 is changed to $v := \max(v, \text{UBScore}(e_S, e'))$.

6. Discussions

In this section, we discuss how to extend our proposals to other distance metrics (Section 6.1) and what properties render a scoring function acceptable to our proposed frameworks (Section 6.2).

6.1. Extension to Other Distances

Our discussions in previous sections are presented in the context of Euclidean space: an object's neighborhood is defined using Euclidean distance, and the algorithms use R-tree and its variants with aggregates. As a matter of fact, our proposals can be extended to work with other types of spatial distances, e.g., spatial network distance and Manhattan distance.

There are several key points in extending our proposals to support the generic distance. First, we need to define the object neighborhood in Definition 1 using the generic distance $dist_g(s, o)$ between spatial objects s and o. Second, we should use the generic distance based neighborhood and the generic distance in defining the score functions (Definitions 2, 4, 5, and 10). Third, we need to use the M-tree [6] and its proper variants to index spatial objects in order to ensure the proposed algorithms still work for the generic distance. According to the definition of M-tree, all objects in an entry e's subtree are within the covering radius $e.cr$ from the routing object $e.ro$. Figure 9 shows two such entries in an M-tree.

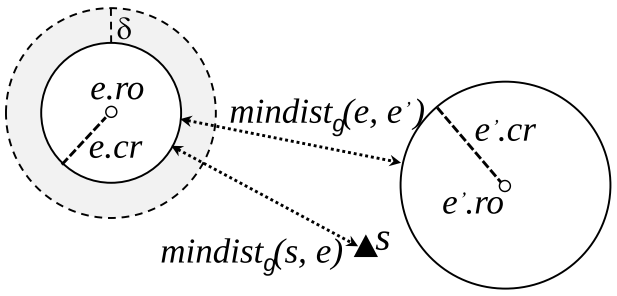


Figure 9: M-tree for Generic Distance

For the different algorithms, the M-tree is used or augmented as follows. In the IS approach for all three score functions (c.f. Table 2), the M-tree is used to index competitor set P. Given an object s and an M-tree subtree entry e, the lower bound distance between them, i.e., $mindist_g(s, e)$, is $dist_g(s, e.ro) - e.cr$, as illustrated in Figure 9. In the BFS and SJB approaches for the dominator counting and distance sensitive score functions, an aggregate `COUNT` M-tree indexes competitor set P and an M-tree indexes candidate object set S. An aggregate M-tree stores an additional *count* value in each of its non-leaf entry to maintain the number of objects in the corresponding subtree. For two M-tree subtree entries e and e', the generalized $mindist_g$ between them is $mindist_g(e, e') = dist_g(e.ro, e'.ro) - e.cr - e'.cr$, as illustrated in Figure 9. In the BFS and SJB approaches for the disadvantage aware score function, sets P and S are indexed by a min-aggregate M-tree and a max-aggregate M-tree, respectively. A min- or max-aggregate M-tree augments each node entry with an additional c-dimensional quality vector in the same way as described in Section 5.3.

6.2. Properties of Score Functions

In this section, we discuss what instances of the generic score function (Definition 2 and Equation 2) can be processed by the query frameworks proposed in this paper.

First, the IS approach is able to process an arbitrary score function that is based on the generic definition because IS processes each spatial object in a given set S in a full iterative manner. In each iteration, IS calculates the current object o's score according to the concrete definition of the score function. Also due to this reason, IS may be inefficient if the score function is time-consuming to calculate.

Second, the BFS approach requires that an upper bound must be derivable from the score function used in the query. Specifically, such an upper bound should be for a subset e_S of the given object set S, where e_S actually corresponds to an entry in the tree that indexes S. Without such an upper bound score, BFS would not be able to prune objects in S by discarding all candidate objects in an unpromising subtree rooted at e_S. In other words, a score function is inapplicable to BFS if it does not support subtree based upper bound estimation. Moreover, a tight upper bound tends to improve the object pruning effect and thus the efficiency of BFS. If a subtree's upper bound score is smaller, it gets a lower priority in subsequent query processing, i.e., a higher chance to be pruned.

Third, the SJB approach also requires that an upper bound must be derivable from the score function used in the query. Again, such an upper bound should be derived for a subtree rooted at entry e_S in the tree index for the given object set S. Likewise, a tight upper bound can help improve the object pruning and the query processing efficiency of SJB.

Also, an upper bound UB as described above should be monotonic such that it can be used by BFS or SJB. Specifically, $UB(e'_S) \leq UB(e_S)$ must hold if $e'_S \subseteq e_S$. In either approach, e_S corresponds to a subtree in the index for object set S and e'_S corresponds to a descendant subtree under e_S. Note that all the upper bounds derived in Sections 3, 4 and 5 are monotonic.

7. Experimental Studies

In this section, we report on experimental studies on our proposed algorithms for querying spatial data with respect to neighborhood dominators.

All algorithms were implemented in Java (JDK-8u151 for Windows x64) and were run on a PC enabled by an Intel 4-Core i5-3570 with 8 GB RAM and Windows 10 64-bit OS. We used real and synthetic data sets for both the competitor set P and the candidate set S. In each data set, all spatial coordinates were normalized to Euclidean space $[0{,}10000]^2$, whereas each quality attribute was normalized to the unit interval [0,1]. The implementation and experiments were disk-based. The disk page size was set to 4K bytes for the data and index files, and an LRU memory buffer with 512K bytes was used for all trees. We issued 20 queries for each test case, and the spatial distance constraint δ in each query was a random value in (0,500] unless s-tated otherwise. We measure the average number of node accesses per query. In all query algorithms, node accesses dominate the total query processing cost; other costs like loading data are relatively very small compared to node accesses and thus are omitted.

7.1. Results on Real Data Sets

In this part of experiments, we used a real data set of the real properties (RPT) in a large city[4]. The data was cleaned as follows. First of all, we removed all those records without longitude and latitude. From the remaining records, we removed those spatial outliers scattered in the border of

[4]As required by the data provider, we are not allowed to disclose the city name.

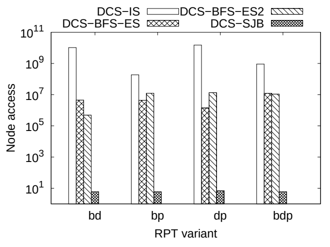
the spatial domain, far from the majority in the center. Subsequently, from the multiple quality attributes we selected three numeric ones with few null values, namely *number of bedrooms*, *distance to the nearest supermarket*, and *price*. A record with a null value on one of these attributes was then removed as well. As a result, we obtained 548,548 records of the schema (*longitude*, *latitude*, *bedroom*, *distance*, *price*). Value conversion was done on a quality attribute if necessary, e.g., a higher bedroom value was converted to a lower value in the normalized range [0, 1]. This way, lower values are preferable to higher ones. Furthermore, we used a sampling to pick one third (182,852 records) as the S data set and the others (365,698 records) formed the P data set. In particular, the normalized spatial domain was divided by a 100 by 100 grid, and one third of the records inside each grid cell were randomly picked into the sample to form the S set.

After the aforementioned steps, we used different quality attribute combinations and obtained four variants of P data set: *bedroom* and *distance* (denoted as bd), *bedroom* and *price* (denoted as bp), *distance* and *price* (denoted as dp), and all three attributes (denoted as bdp). The corresponding S data set variants were obtained in the same way.

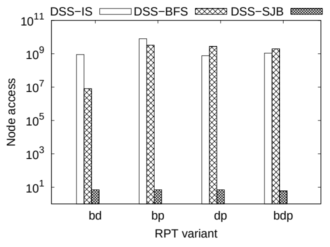
First, we used a random δ in each query for all three score functions (DCS, DSS, and DAS). The results are shown in Figure 10. For all the three functions, SJB approach is significantly more efficient than all others, BFS is the second and IS is the worst. SJB's superiority is attributed to its join nature—it processes node entries that index objects in S only with relevant node entries that index P. In contrast, IS approach processes each object in S, issuing a recursive range search via the R-tree that indexes P. BFS approach's performance is between SJB and IS. BFS outperforms IS because BFS does not process objects in S sequentially. Instead, BFS processes groups of objects in S via an R-tree; however, its pruning of tree nodes is considerably less aggressive than SJB. According to Figure 10(a), Algorithm 6 does not always improve Algorithm 3 in BFS approach using DCS. The price attribute in our real data has the largest domain and the most diverse values in the domain, which makes it more difficult to achieve a better overestimate of the counting based upper bounds used by BFS.

Next, we used the bdp data set and varied distance δ from 100 to 500. In our setting, these δ values mean a neighborhood radius of 1% to 5% the width of the spatial domain, and they form big search regions in the large city in our setting. The results are shown in Figure 11. Again, SJB approach outperforms all others in all test cases, and IS is still the worst.

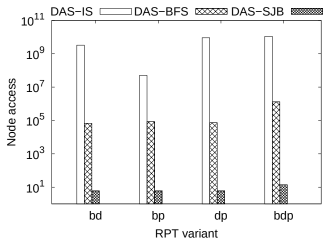
More importantly, for all score functions, SJB approach is very steady as δ increases. Although a larger δ leads to a larger neighborhood and thus more objects from P to process, it does not worsen SJB that works in a join fashion. On the one hand, more objects from P may not considerably increase the number of P's tree nodes for SJB to process. On the other hand, the upper bounds and the pruning techniques used by SJB work effectively in ruling out unpromising tree nodes and thus objects.



(a) DCS

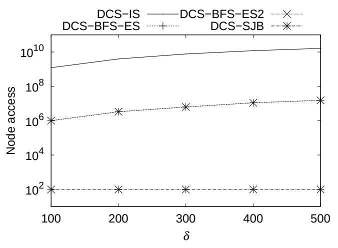


(b) DSS

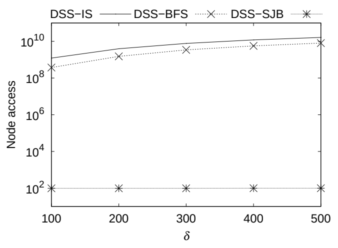


(c) DAS

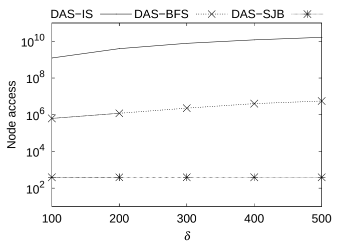
Figure 10: Results on Real Data Sets



(a) DCS



(b) DSS



(c) DAS

Figure 11: Effect of δ on Real Data Sets

7.2. Results on Synthetic Data Sets

We also tested the scalability of our approaches using synthetic data sets. In particular, we investigated the effects of $|P|$, $|S|$, c (the number of quality attributes), and the distance δ. The settings of parameters are given in Table 3 where bold fonts indicate the default settings. For all spatial objects, the quality attributes follow independent (IN) and anti-correlated (AC) distributions in their normalized unit domain $[0,1]^c$. The quality attributes of a particular distribution were generated according to a previous work [2]. The experimental results for the IN distributions are similar to their counterparts for the AC distributions, and therefore we only show the results for the latter.

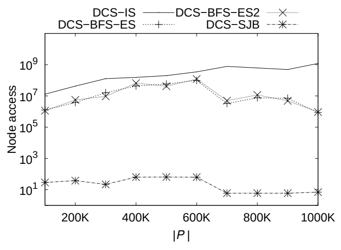
Parameter	**Setting**
$\|P\|$	100K, 200K, ..., **1000K**
$\|S\|$	**10%**$\cdot\|P\|$, 20%$\cdot\|P\|$, ..., 60%$\cdot\|P\|$
c	**2**, 3, 4, 5

Table 3: Parameters of synthetic data sets

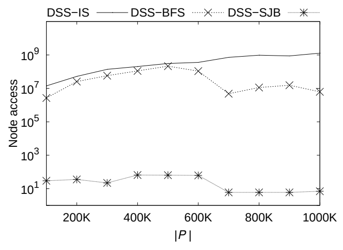
First, we varied $|P|$ from 100K to 1000K and used the default settings for the other parameters. The results are shown in Figure 12(a), (b) and (c) for DCS, DSS and DAS, respectively. Overall, SJB approach performs the best in all settings, BFS approach the second, and IS approach the worst. With score functions DCS and DSS, BFS and SJB's node access cost after $|P| =$ 700K is lower than that when P is smaller. For the four largest P sets, it is found that some objects in the corresponding S sets get many more neighborhood dominators, partly due to the higher density in the space domain. Consequently, after their concrete DCS and DSS scores are computed, such objects are exploited by BFS and SJB to prune more unpromising tree nodes in query processing. In contrast, IS is not affected in such cases since it has to process all objects iteratively without pruning. On the other hand, for BFS and SJB with DAS function, we do not observe the same performance improvement as with DCS and DSS functions. We attribute the difference to the fact that DAS function is not based on counting the number of neighborhood dominators for objects in S.

Next, we fixed $|P|$ to 1000K and varied $|S|$ from 10% to 50% of $|P|$. The results are shown in Figure 13(a), (b) and(c) for DCS, DSS and DAS, respectively. SJB approach still outperforms the alternatives for all three score functions. Overall, SJB is also the most scalable as a larger S does not incur much more node accesses for SJB.

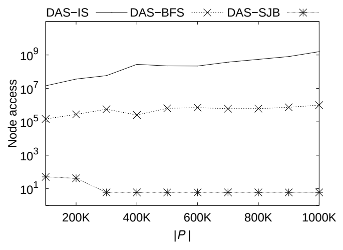
Moreover, we varied the number of quality attributes (c) from 2 to 5 and used the default values for the others. The results are shown in Figure 14(a), (b) and (c) for DCS, DSS and DAS, respectively. SJB approach still incurs the least node accesses; it is the most scalable for all three score functions. As c increases, IS approach incurs less node accesses. This is because more quality attributes tend to result in less neighborhood dominators for an object in S, which results in an overall effect that makes the sequential scanning based query processing slightly faster with all three score functions.



(a) DCS

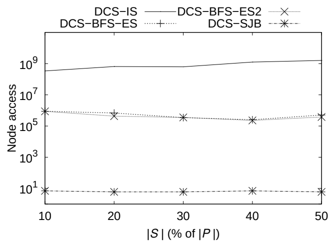


(b) DSS

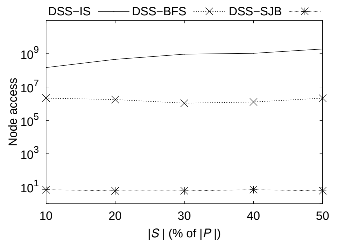


(c) DAS

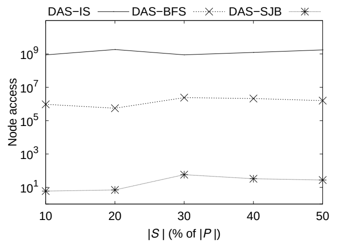
Figure 12: Effect of $|P|$ on Synthetic Data Sets



(a) DCS



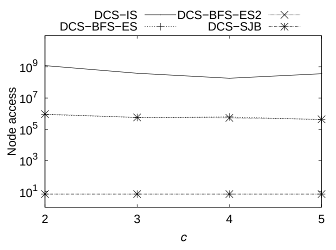
(b) DSS



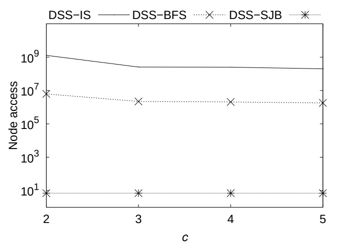
(c) DAS

Figure 13: Effect of $|S|$ on Synthetic Data Sets

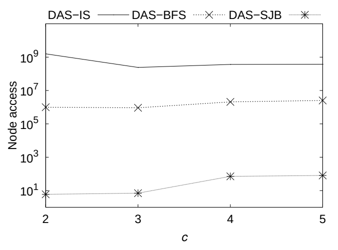
Finally, we varied δ from 100 to 500 and used the default settings for the others. The results are shown in Figure 15(a), (b) and (c) for DCS, DSS and DAS, respectively. As a neighborhood becomes larger, more dominators are likely to be included. This explains the overall increase in the node accesses, especially for IS and BFS. Nevertheless, SJB is still the most efficient and almost insensitive to the neighborhood size.



(a) DCS

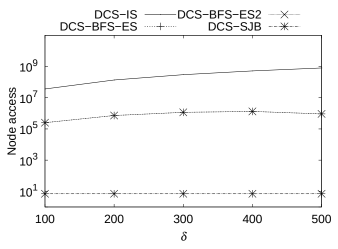


(b) DSS

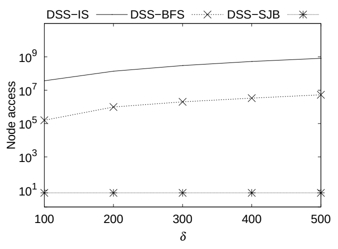


(c) DAS

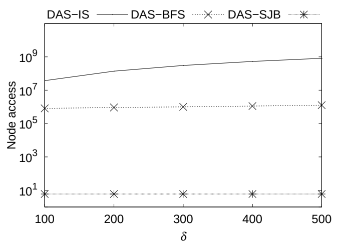
Figure 14: Effect of c on Synthetic Data Sets



(a) DCS



(b) DSS



(c) DAS

Figure 15: Effect of δ on Synthetic Data Sets

8. Related Work

In this section, we review related work on location selection queries (Section 8.1), skyline queries involving spatial data (Section 8.2), spatial keyword queries (Section 8.3), and spatial join (Section 8.4).

8.1. Location Selection Queries

Optimal location selection query is a well studied topic. Du et al. [8] 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. [32] formulated a different top-k most influential spatial sites query, which returns k sites (from S and within Q) having the highest influences. Given an additional location set Q, Zhang et al. [37] proposed the min-dist optimal-location query that selects from Q a location l that results in the minimal average distance from every object in O to its nearest site in $S \cup \{l\}$. Xiao et al. [33] studied location selection queries in road networks where network distances are used to define optimal locations. Unlike the MEO query studied in this paper, these location selection queries do not consider quality attributes associated with spatial objects.

Given two spatial object sets O and P, a bichromatic reverse k nearest neighbor (BRkNN) query [13] requires a query object $p \in P$ and returns all the objects from O that have p as one of its k nearest neighbors in P. A maximizing BRkNN (MaxBRkNN) query [30, 29, 40] returns the optimal region such that if an object p is placed in the region the cardinality of p's BRkNN query result in O is maximized, i.e., the number of objects from

O having p as a nearest neighbor is the highest. Unlike our MEO query, BRkNN and MaxBRkNN queries only consider spatial locations and ignore non-spatial quality attributes associated with spatial objects.

Yiu et al. [35, 36] formalized the top-k spatial preference query that requires a location set $\mathcal{D}$ and several feature object set $\mathcal{F}_1, \ldots, \mathcal{F}_m$. Each $\mathcal{F}_i$ set covers a particular type, e.g., restaurants. Each feature object in an $\mathcal{F}_i$ set is associated with a location and a single quality value. The query returns the k locations from $\mathcal{D}$ with the highest aggregate scores derived from the m best quality values, each from a corresponding feature object set, in the location's spatial proximity. The basic setting of our MEO query is apparently different in that an MEO query requires two spatial objects sets: one for candidates and one for potential threats (i.e., possible dominators). Also, the score functions of MEO queries involve a candidate object's location and quality attributes as well, whereas the score functions used in top-k spatial preference queries [35, 36] do not support the multi-dimensional dominance relationship used in MEO queries. Furthermore, the score functions in [35] do not integrate the difference in distance; the extended score functions in [36] integrate a distance decay effect without the use of neighborhood as in our MEO query.

Rocha-Junior et al. [24] proposed a materialization technique that stores pairs of distance and feature quality value to speed up the processing of top-k spatial preference queries [35, 36]. However, the proposed technique is inapplicable to MEO queries mainly for two reasons. First, the two problem formulations are very different as described above. Second, in the setting of MEO queries there is no counterpart for the originally available feature quality values used by the materialization technique. In contrast, each score in MEO queries must be computed on the fly through multiple steps including distance restriction, dominance check, and calculation according to the concrete score function. As a result, it is not guaranteed that the materialized objects are sufficient for finding the top-k objects according to the score functions used in MEO queries.

Focusing on spatial objects with both locations and quality attributes, Li et al. [15] defined three location selection problems by combining dominance relationship and spatial distance. First, given a spatial object q from set O, a nearest dominator query returns q's nearest neighbor in O that dominates q in terms of quality attributes. Second, given a hyperplane P in the non-spatial attribute space of O, a least dominated, profitable points query returns object(s) o from O such that o is dominated by some object(s) on P

and the spatial distance from o to its dominator is maximized. Third, given a hyperplane P as above, a minimal loss and least dominated points query returns object(s) o from O such that its spatial distance to its dominator is not less than a threshold δ and its distance to P on all non-spatial attributes is minimized. All these three problems only consider one data set and their objectives to minimize or maximize are different from the score functions in our MEO query. Therefore, the solutions from [15] cannot be applied to solve our MEO query.

Wen et al. [28] defined continuous relational top-k query (CRTQ) to continuously return the k dynamic objects having the highest scores with respect to a group of related objects. A group function F_G decides whether two given objects are related or not, whereas a score function F_S determines an object's score with respect to a given group of objects. There are important differences between CRTQ and MEO queries. First, CRTQ accepts a single relation of objects, whereas the MEO query works for two spatial object sets. Second, CRTQ and MEO queries assume different data models. CRTQ works for dynamic data where objects change their attribute values continuously, while MEO query works for static spatial objects with fixed locations and quality attributes. Objects in CRTQ are not necessarily spatial objects. When spatial objects are considered in CRTQ, their coordinates are treated equally as non-spatial attributes. In contrast, MEO query differentiates spatial attributes and quality attributes in defining object scores. Third, CRTQ and MEO queries use different score functions. A concrete MEO query only needs one score function, while CRTQ employs two. Although the group function F_G is abstract enough to capture if an object is in another's neighborhood, it is unclear how the score function F_S can support the dominance based scores considered in MEO query.

Our early work [19] finds from a set of locations the one yielding the longest distance from the dominator in a set of objects when a fixed quality vector is to be assigned to candidate locations. In contrast, this paper handles the MEO query on two sets of spatial objects, i.e., finding the top-k objects in one set that have the highest scores with respect to neighborhood dominators from the other set. Although both works employ the three algorithmic frameworks (IS, BFS, SJB), this paper's algorithms contain local designs specialized for the MEO query. Furthermore, this paper conducts thorough analysis on the three score functions proposed, and devises effective search bounds that enable us to adapt the BFS and SJB algorithmic frameworks to solve the MEO query with new score functions.

8.2. Skyline Queries Involving Spatial Data

Skyline queries has been extended to involve spatial data in different scenarios. Huang and Jensen [11] proposed an in-route skyline query for location-based services. When moving in a pre-defined road route towards a 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 [25] studied the spatial skyline query, a specialized version of the dynamic skyline query [22]. 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. [12] 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. [39] addressed how to compute the valid scope for such a query result without knowing the movement pattern of the object.

Given a set of spatial objects and a set of locations, Shi et al. [26] formulated a 2-dimensional skyline query that returns optimal locations with high influence and low cost. A location's influence is defined as the number of neighboring spatial objects in its proximity, whereas its cost is derived from its quality attributes.

These works above are similar in that they all study skyline problems that involve multiple dimensions, i.e., spatial distances and/or those derived from multiple quality attributes. The main difference of the MEO query is that the MEO query itself is not a skyline problem. Recalling the motivation example in Section 1, 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.

Xie et al. [34] proposed to rank spatial objects according to their dominating capability, which is opposite to this paper's quantifying the effect of dominators in neighborhood. Moreover, work [34] handles a single data set whereas MEO query requires two sets as input.

Unlike the constrained skyline query [22] where objects are filtered by a constraint region in the domain of quality attributes, the spatial distance constraint δ employed in the MEO query is only used in the spatial domain but not on quality attributes.

8.3. Spatial Keyword Queries

Spatial keyword queries combine geo-spatial queries and keyword search, enabling search for interesting locations that are associated with relevant keywords. Multiple geo-textual indexes have been proposed for processing s-patial keyword queries. Spatial-first Index (ST) and text-first Index (TS) [27] are two grid based geo-textual indexing schemes for searching geo-tagged web documents. The integrated inverted index (I^3) [38] associates keywords to quadrants in the quadtree structure for the spatial domain. The KR*-Tree (Keyword R*-tree) [10] augments each R*-tree node with the set of keywords that appear in its subtree. The IR^2-tree [9] augments each R-tree node with a signature file, a bitmap that is the union of all signatures of its subtrees, to describe the keywords in that node. The IR-tree and its variants [7, 16, 31] are essentially an R-tree in which each node is augmented with an inverted index describing the textual information in the subtree. The Spatial Invert-ed Index (S2I) [23] maps each frequent keyword to an aggregated R-tree and each less frequent keyword to a block in a file. Chen et al. [5] conducted intensive experiments on typical geo-textual hybrid indexes to evaluate their performance for different types of queries. In contrast to these geo-textual indexes, the augmented R-trees used in our algorithms involve object counts and quality attributes rather than keywords or other textual information.

8.4. Spatial Join

Given a distance δ, and two spatial data sets S and P, *the δ-distance join* returns each pair $\langle s, p \rangle$ ($s \in S$ and $p \in P$) such that their Euclidean distance $dist(s, p)$ is less than δ. The *R-tree join* (RJ) [3] can be applied to evaluate the δ-distance join if 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) satisfy $mindist(e_S, e_P) \leq \delta$, 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 [14]. Zhu et al. [41] proposed the *top-k spatial join* for computing k objects of S that intersect the largest number of objects in P. These studies consider only spatial locations and spatial relationships between them; they do not consider quality attributes as MEO query does.

9. Conclusion

In this paper, we formalize the most endangered object (MEO) query for spatial data. Given a competitor object set P, a candidate object set S, and a distance δ, the MEO query returns from S an object s with the maximum effect indicated by the dominators from its δ-neighborhood in P. For this generic query, we define three score functions, namely the *dominator counting score*, the *distance sensitive score*, and the *disadvantage aware score* to quantify the neighborhood dominators' effect.

We propose three approaches for processing the MEO query. The IS approach is an iterative search that only requires P to be indexed by an R-tree. The best-first search (BFS) approach additionally requires an aggregate R-tree for S, and prunes by using the aggregate counts in tree nodes. The spatial-join based approach (SJB) joins two R-trees and prunes tree n-odes more aggressively. We derive appropriate upper bounds for each score function, such that each approach is able to accommodate all three score functions. We also discuss how to support the generic distance in the MEO query and what score function properties are needed by the three query processing approaches. We conduct extensive experimental studies on both real and synthetic data sets. The experimental results disclose that the SJB approach outperforms others in all settings and it is very scalable and stable under different settings.

References

[1] C. Böhm. A cost model for query processing in high dimensional data spaces. *ACM Trans. Database Syst.*, 25(2):129–178, 2000.

[2] S. Borzonyi, D. Kossmann, and K. Stocker. The skyline operator. In *Proc. ICDE*, pages 421–430, 2001.

[3] T. Brinkhoff, H.-P. Kriegel, and B. Seeger. Efficient processing of spatial joins using r-trees. In *Proc. SIGMOD*, pages 237–246, 1993.

[4] A. R. Butz. Alternative Algorithm for Hilbert's Space-Filling Curve. *IEEE Trans. Comput.*, C-20(4):424–426, 1971.

[5] L. Chen, G. Cong, C. S. Jensen, and D. Wu. Spatial keyword query processing: An experimental evaluation. *PVLDB*, 6(3):217–228, 2013.

[6] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Proc. VLDB*, pages 426–435, 1997.

[7] G. Cong, C. S. Jensen, and D. Wu. Efficient retrieval of the top-k most relevant spatial web objects. *PVLDB*, 2(1):337–348, 2009.

[8] Y. Du, D. Zhang, and T. Xia. The optimal-location query. In *Proc. SSTD*, pages 163–180, 2005.

[9] I. D. Felipe, V. Hristidis, and N. Rishe. Keyword search on spatial databases. In *Proc. ICDE*, pages 656–665, 2008.

[10] R. Hariharan, B. Hore, C. Li, and S. Mehrotra. Processing spatial-keyword (sk) queries in geographic information retrieval (gir) systems. In *Proc. SSDBM*, page 16, 2007.

[11] X. Huang and C. S. Jensen. In-route skyline querying for location-based services. In *Proc. W2GIS*, pages 120–135, 2004.

[12] Z. Huang, H. Lu, B. C. Ooi, and A. K. H. Tung. Continuous skyline queries for moving objects. *TKDE*, 18(12):1645–1658, 2006.

[13] F. Korn and S. Muthukrishnan. Influence sets based on reverse nearest neighbor queries. In *Proc. SIGMOD*, pages 201–212, 2000.

[14] N. Koudas and K. C. Sevcik. High dimensional similarity joins: Algorithms and performance evaluation. In *Proc. ICDE*, pages 466–475, 1998.

[15] C. Li, A. K. H. Tung, W. Jin, and M. Ester. On dominating your neighborhood profitably. In *Proc. VLDB*, pages 818–829, 2007.

[16] Z. Li, K. C. K. Lee, B. Zheng, W.-C. Lee, D. L. Lee, and X. Wang. IR-tree: An efficient index for geographic document search. *IEEE Trans. Knowl. Data Eng.*, 23(4):585–599, 2011.

[17] H. Lu and C. S. Jensen. Upgrading uncompetitive products economically. In *Proc. ICDE*, pages 977–988, 2012.

[18] H. Lu and M. L. Yiu. Identifying the most endangered objects from spatial datasets. In *Proc. SSDBM*, pages 608–626, 2009.

[19] H. Lu and M. L. Yiu. On computing farthest dominated locations. *IEEE Trans. Knowl. Data Eng.*, 23(6):928–941, 2011.

[20] B. Moon, H. V. Jagadish, C. Faloutsos, and J. H. Saltz. Analysis of the Clustering Properties of the Hilbert Space-Filling Curve. *IEEE Trans. Knowl. Data Eng.*, 13(1):124–141, 2001.

[21] D. Papadias, P. Kalnis, J. Zhang, and Y. Tao. Efficient olap operations in spatial data warehouses. In *Proc. SSTD*, pages 443–459, 2001.

[22] D. Papadias, Y. Tao, G. Fu, and B. Seeger. An optimal and progressive algorithm for skyline queries. In *Proc. SIGMOD*, pages 467–478, 2003.

[23] J. B. Rocha-Junior, O. Gkorgkas, S. Jonassen, and K. Nørvåg. Efficient processing of top-k spatial keyword queries. In *Proc. SSTD*, pages 205–222, 2011.

[24] J. B. Rocha-Junior, A. Vlachou, C. Doulkeridis, and K. Nørvåg. Efficient processing of top-k spatial preference queries. *PVLDB*, 4(2):93–104, 2010.

[25] M. Sharifzadeh and C. Shahabi. The spatial skyline queries. In *Proc. VLDB*, pages 751–762, 2006.

[26] J. Shi, H. Lu, J. Lu, and C. Liao. A skylining approach to optimize influence and cost in location selection. In *Proc. DASFAA Part II*, pages 61–76, 2014.

[27] S. Vaid, C. B. Jones, H. Joho, and M. Sanderson. Spatio-textual indexing for geographical search on the web. In *Proc. SSTD*, pages 218–235, 2005.

[28] J. Wen, V. J. Tsotras, and D. Zhang. On continuously monitoring the top-k moving objects with relational group and score functions. *SIGSPATIAL Special*, 1(3):5–10, 2009.

[29] R. C. Wong, M. T. Özsu, A. W. Fu, P. S. Yu, L. Liu, and Y. Liu. Maximizing bichromatic reverse nearest neighbor for L p -norm in two- and three-dimensional spaces. *VLDB J.*, 20(6):893–919, 2011.

[30] R. C. Wong, M. T. Özsu, P. S. Yu, A. W. Fu, and L. Liu. Efficient method for maximizing bichromatic reverse nearest neighbor. *PVLDB*, 2(1):1126–1137, 2009.

[31] D. Wu, G. Cong, and C. S. Jensen. A framework for efficient spatial web object retrieval. *VLDB J.*, 21(6):797–822, 2012.

[32] T. Xia, D. Zhang, E. Kanoulas, and Y. Du. On computing top-t most influential spatial sites. In *Proc. VLDB*, pages 946–957, 2005.

[33] X. Xiao, B. Yao, and F. Li. Optimal location queries in road network databases. In *Proc. ICDE*, pages 804–815, 2011.

[34] X. Xie, H. Lu, J. Chen, and S. Shang. Top-k neighborhood dominating query. In *Proc. DASFAA Part I*, pages 131–145, 2013.

[35] M. L. Yiu, X. Dai, N. Mamoulis, and M. Vaitis. Top-k spatial preference queries. In *Proc. ICDE*, pages 1076–1085, 2007.

[36] M. L. Yiu, H. Lu, N. Mamoulis, and M. Vaitis. Ranking spatial data by quality preferences. *IEEE Trans. Knowl. Data Eng.*, 23(3):433–446, 2011.

[37] 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.

[38] D. Zhang, K.-L. Tan, and A. K. H. Tung. Scalable top-k spatial keyword search. In *Proc. EDBT*, pages 359–370, 2013.

[39] B. Zheng, K. C. K. Lee, and W.-C. Lee. Location-dependent skyline query. In *Proc. MDM*, pages 148–155, 2008.

[40] Z. Zhou, W. Wu, X. Li, M. Lee, and W. Hsu. Maxfirst for maxbrknn. In *Proc. ICDE*, pages 828–839, 2011.

[41] M. Zhu, D. Papadias, J. Zhang, and D. L. Lee. Top-k spatial joins. *IEEE Trans. Knowl. Data Eng.*, 17(4):567–579, 2005.