On Computing Farthest Dominated Locations

Hua Lu, Member, IEEE, Man Lung Yiu

Abstract—In reality, spatial objects (e.g., hotels) not only have spatial locations but also have quality attributes (e.g., price, star). An object \( p \) is said to dominate another one \( p' \), if \( p \) is no worse than \( p' \) with respect to every quality attribute and \( p \) is better on at least one quality attribute. Traditional spatial queries (e.g., nearest neighbor, closest pair) ignore quality attributes, whereas conventional dominance-based queries (e.g., skyline) neglect spatial locations. Motivated by these observations, we propose a novel query by combining spatial and quality attributes together meaningfully. Given a set of (competitors’) spatial objects \( P \), a set of (candidate) locations \( L \), and a quality vector \( \Psi \) as design competence (for \( L \)), the farthest dominated location query (FDL) retrieves the location \( s \in L \) such that the distance to its nearest dominating object in \( P \) is maximized. FDL queries are suitable for various spatial decision support applications such as business planning, wild animal protection, and digital battle field systems. As FDL queries are not solved by existing techniques, we develop several efficient R-tree based algorithms for processing FDL queries, which offer users a range of selections in terms of different indexes available on the data. We also generalize our methods to support the generic distance metric and other interesting query types. The experimental results on both real and synthetic datasets disclose the performance of those algorithms, and identify the most efficient and scalable one among them.

Index Terms—Spatial Objects, Query Processing, Database Management.

1 INTRODUCTION

Spatial objects (e.g., hotels [1]) in reality are associated with multiple quality attributes (e.g., price, star), in addition to their spatial locations. Traditional spatial queries and joins (e.g., nearest neighbor [12], closest pair [10]) focus on manipulating only spatial locations and distances, but they ignore the importance of quality attributes.

The dominance comparison is suitable for comparing two objects with respect to multiple quality attributes. For the sake of simplicity, we assume that the domain of each quality attribute is fully ordered (e.g., integer domain). An object \( A \) is said to dominate another object \( B \), if \( A \) is no worse than \( B \) for all quality attributes and \( A \) is better than \( B \) for at least one quality attribute. The skyline query [5], [22], [15], [8], [18], built upon the dominance comparison, retrieves the objects that are not dominated by any other. However, the skyline query neglects the significance of spatial locations.

In practice, spatial data analysts are interested in combining both distance and dominance comparison to find results satisfying their specific applications. Consider the example that a hotel chain is planning to open a new hotel in a metropolitan city. The city already has several existing hotels (as competitors), each associated with its location and quality values. The new hotel will be built such that its quality values reach the design competence.

Among a predefined set of candidate locations for the new hotel, a candidate location is desired if it is far away from its nearest existing hotel that dominates its design competence. This way, the new hotel will not be in a disadvantaged position in business competition with other hotels within its proximity.

Figure 1 shows the locations of a set of competitors’ hotels (dots) and a set of candidate locations (triangles) for building the new hotel. Candidate locations can be obtained in different ways, e.g., from urban planning department or from private sectors. The quality attributes of existing hotels are listed in Figure 2a, where lower price and higher star values are preferable. Suppose that the hotel chain wishes the new hotel to be at 4-star quality with $200 room price, i.e., the design competence of the new hotel is \( \Psi = (200, \star 4) \).

Unfortunately, both the skyline query and traditional spatial queries fail to find a desirable location for build-
ing the new hotel. By issuing a skyline query on the quality attributes of hotels in Figure 1, we obtain the result set \{h_1, h_2, h_3\}, which however offers no recommendation at all regarding the candidate locations \{s_1, s_2, s_3, s_4\}. Therefore, the skyline query is unable to select an appropriate location for building the new hotel.

Existing spatial queries are also not helpful here. Let \(\text{dist}(s_i, h_j)\) denotes the Euclidean distance between a location \(s_i\) and a hotel \(h_j\). The nearest neighbor (NN) of \(s_i\) is the hotel \(h_j\) having the minimum value of \(\text{dist}(s_i, h_j)\) \[1\]. Figure 2b shows the NN of each \(s_i\). One may suggest finding the location \(s_i\) (for the new hotel) such that its distance to its NN is maximized; however, this totally ignore quality attributes of the hotels.

Similar needs are also seen in other spatial decision support applications. In wild animal rehabilitation [3], an appropriate location is selected from a set of options for returning an animal to the nature, after its treatment in a rehabilitation center. A location is preferred if it is far away from (competing) animals with multiple better abilities (e.g., speed, weight, age). Since those animals are much stronger in fighting for essential resources, e.g., water and food, and hence they will endanger the rehabilitated animal that is still unaccustomed to wildlife. By maximizing the distances to potential nearest dominators, the preferred location improves the survival chance of the rehabilitated animal in the nature.

In an example of airborne landing (in a digital battle field system [2]), serious threats come from nearby locations where enemies have multiple advantages (e.g., better equipment, more soldiers). Among the set of candidate locations for landing, the one with the largest distance to its nearest threat is desired, in order to enhance the safety of landing.

Motivated by these real-life examples, in this paper we combine both spatial locations and quality attributes to define a novel query that retrieves practically meaningful locations as expected in the above examples.

### 1.1 Problem Statement

Let \(c\) be the number of (numeric) quality attributes. A quality vector is a point \(\psi\) in the \(c\)-dimensional space \(\mathbb{R}^c\), where each dimension refers to a quality attribute. As a shorthand notation, we use \(\psi[i]\) to represent the \(i\)-th (quality) attribute value of \(\psi\).

The notion of dominance [5] is used to compare quality vectors. A quality vector \(\psi\) is said to dominate another one \(\psi'\) (denoted as \(\psi \prec \psi'\)), if \(\exists 1 \leq i \leq c\), \(\psi[i]\) is better than \(\psi'[i]\) and \(\forall 1 \leq i \leq c\), \(\psi[i]\) is not worse than \(\psi'[i]\).

A location is a pair \((x, y)\) in the Euclidean space \(\mathbb{R}^2\), where \(x\) and \(y\) are coordinate values. A spatial object \(o = (\text{loc}, \psi)\) consists of both a location \(\text{loc}\) and a quality vector \(\psi\). The notation \(\text{dist}(o, o')\) 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 be a dominator of \(o'\) when \(o, \psi \prec o', \psi\).

We proceed to present the definitions of the nearest dominator (ND) and nearest dominator distance (n\(\text{dd}\)) [16] of a location \(s\), as follows.

**Definition 1: (Nearest Dominator, Nearest Dominator Distance)** Given a location \(s\), its quality vector \(\psi\), and a set of spatial objects \(P\), the nearest dominator of \(s\) in \(P\) is defined as

\[
\text{ND}(s, \Psi, P) = \arg\min_{o \in P, o, \psi \leq \psi} \text{dist}(s, o)
\]

i.e., the nearest neighbor of \(s\) in \(P\) among those that dominate \(\psi\).

The nearest dominator distance \(\text{n\(\text{dd}\)}(s, \Psi, P)\) of \(s\) is then defined as:

\[
\text{n\(\text{dd}\)}(s, \Psi, P) = \text{dist}(s, \text{ND}(s, \Psi, P)).
\]

Refer to the example in Figures 1 and 2. The ND of \(s_i\) is the hotel \(h_j\) that minimizes the \(\text{dist}(s_i, h_j)\) value, among those hotels dominating the design competence \(\Psi\). Figure 2b lists the NN and ND of each location \(s_i\). It is important to note that NN is not necessarily the same as ND. For example, the NN of \(s_2\) is \(h_4\) which however does not dominate \(s_2\) with respect to its design competence. Whereas its next nearest neighbor \(h_3\) does, which exactly is \(s_2\)'s ND. It is also noteworthy that a location’s ND is not necessarily a skyline point, as indicated by \(h_3\) here. By considering the distance of each location \(s_i\) from its ND, we pick the largest one (i.e., \(\text{dist}(s_3, h_5)\)), and take its location (i.e., \(s_3\)) as the result location for building the new hotel.

Specifically, we define the novel farthest dominated location query as follows.

**Definition 2: (Farthest Dominated Location Query)** Given a set of (competitors’) spatial objects \(P\), a set of (candidate) locations \(L\), and a quality vector \(\Psi\) as the design competence, the farthest dominated location query (FDL) \(^1\) returns from \(L\) a location \(s\) such that the distance \(\text{n\(\text{dd}\)}(s, \Psi, P)\) is maximized, i.e.,

\[
\forall s' \in L, \text{n\(\text{dd}\)}(s, \Psi, P) \geq \text{n\(\text{dd}\)}(s', \Psi, P)
\]

Refer to the hotel example in Figures 1 and 2. There are \(c = 2\) quality attributes (i.e., price and star). The set of objects is \(P = \{h_1, h_2, \ldots, h_6\}\) and the set of locations is \(L = \{s_1, s_2, \ldots, s_4\}\). Hotel \(h_1\) is a spatial object, with a fixed location in the Euclidean space and the quality vector \(h_1, \psi = (180, 4)\). Let the design competence be \(\Psi = (200, 4)\). Location \(s_3\) is the farthest dominated location and its nearest dominator is \(h_5\).

### 1.2 Technical Contributions and Paper Organization

The processing of the FDL query raises non-trivial challenges. First, the ND of a location \(s\) is not necessarily the NN of \(s\), because the NN may not have quality attributes that dominate \(s\) (See Figure 2b). Although it is possible to solve the FDL query using incremental NN queries, as to be detailed in Section 3.1, such a straightforward solution is not efficient because it makes little use of non-spatial quality attributes in designing efficient query processing.

Second, the ND of a location \(s\) is not necessarily in the skyline of all locations in \(P\) with respect to their quality

---

1. The definition of the FDL query can be extended to return the \(k\) farthest dominated locations.
attributes. If an object \( o \) in \( P \) dominates \( s \), it can still be dominated by some other objects in \( P \) and therefore it is not in the skyline. This is exemplified by \( h_3 \) and locations \( s_1 \) and \( s_2 \) in Figures 1 and 2. On the other hand, even if object \( o \) is in the skyline of \( P \), it is not necessary for \( o \) to be the ND of a location \( s \). There are two reasons for this: (1) \( o \) is unable to dominate \( s \) (e.g., \( h_2 \) and \( h_6 \) in the example); (2) another object \( o' \) nearer to \( s \) dominates \( s \) (e.g., \( h_5 \) to \( s_2 \) in the example).

As a result, there is no deterministic relationship between the result of a FDL query and that of a skyline query. Particularly, a FDL query cannot be answered by obtaining the skyline of \( P \) followed by some postprocessing on the skyline. Refer to the example again. Taking into account all objects in \( P \), the FDL is \( s_3 \) and its ND is \( h_5 \). Whereas, if only skyline points in \( P \) are considered, namely \( \{h_1, h_2, h_6\} \), \( s_4 \) will be the FDL and its ND is \( h_1 \). However, this is not desired because the existing competence from non-skyline points is totally ignored.

We in this paper make the following major contributions. First, to the best of our knowledge, this paper is the first to formulate the FDL query that captures practical needs involving not only spatial locations but also quality attributes. Second, we adapt the incremental NN search algorithm to process the FDL query. This makes it possible to find the FDL on legacy implementations without much additional investment. Third, we design specific and more efficient methods for the FDL query. Fourth, we conduct a thorough theoretic analysis on the performance of proposed methods. Fifth, we generalize our proposals to deal with the generic distance metric and other interesting query types. Finally, we conduct an extensive experimental study for the proposed methods on both real and synthetic datasets, and show that our best algorithm is indeed efficient and scalable.

The rest of this paper is organized as follows. Section 2 reviews related work. Section 3 elaborates on our FDL query algorithms. Section 4 covers the technical generalizations. Section 5 experimentally evaluates the proposals, followed by conclusion in Section 6.

2 Related Work

Location selection queries. In the past, different constraints have been combined with conventional spatial queries to select semantically optimal locations or objects. Du et al. [11] proposed the optimal-location query. Involving a site set \( S \), a weighted object set \( O \), and a spatial region \( Q \), an optimal-location query returns a location in \( Q \) with maximum influence. The influence of a location \( l \) is the total weights of objects in \( O \), each of which has \( l \) as its nearest neighbor in \( S \cup \{l\} \). In other words, the influence of a location is the sum of weights of all its reversed nearest neighbors (RNNs). With the same influence definition, Xia et al. [24] defined a different top-\( t \) most influential spatial sites query, which returns \( t \) sites (from \( S \) and within \( Q \)) with the largest influences. Within the same context, Zhang et al. [26] proposed the min-dist optimal-location query. Different from maximizing influence, such a query selects from \( Q \) a location \( l \) which minimizes the average distance from every object in \( O \) to its nearest site in \( S \cup \{l\} \). Such location-optimal queries differ from our FDL query in that they do not consider multi-dimensional dominance relationship among possible locations and existing objects. This makes their solutions inapplicable to our specific problem.

Yiu et al. [25] formalized the top-\( k \) spatial preference query, which returns the \( k \) spatial objects with the highest ranking scores. Objects are ranked based on an aggregate score function that is defined for the feature qualities in their spatial proximity. Such score functions, however, do not support multi-dimensional dominance relationship. Therefore, the top-\( k \) spatial preference query is essentially different from our FDL query.

Li et al. [16] defined the concepts of ND and ndd, combined dominance relationship with spatial distance, and defined complex location selection problems. Two fundamental differences distinguish our work from that one. First, only one set is considered in [16], from which desirable objects are selected. Whereas our problem involves two datasets with different practical semantics, and selects best locations from the location set. Second, a linear restriction on quality attributes is used in [16]. The distance from an object to that restriction within the quality attributes space is to be minimized. Whereas our problem employs a design competence vector, and the Euclidean distance from a location to its nearest dominator is to be maximized. Because of the different constraints and contrary optimization objectives, our problem in this paper is not a bichromatic version of the problem in [16]. As a result, the approaches in [16] cannot be directly applied to solve the problem in this paper. Also, Li et al. [16] have not studied the problem for generic distance metric like the road network distance or Manhattan distance. In contrast, we develop technical solutions for such generic cases in this paper.

Skylining in spatial and spatiotemporal context. Dominance relationship has been adopted in spatial and spatiotemporal database to define specific problems. Huang and Jensen [13] proposed a in-route skyline query for location-based services. When moving along a pre-defined road route towards her/his destination, a user may visit points of interest in the network. Points to visit are selected in terms of multiple distance-related preferences like detour and total traveling distance. The authors optimize such selections using skyline queries involving specific interesting dimensions. Sharifzadeh and Shahabi [20] defined a spatial skyline query (SSQ) in spatial databases. Given a set of query points \( Q = \{q_1, \ldots, q_n\} \) and two points \( p \) and \( p' \), \( p \) is said to spatially dominates \( p' \) iff \( \text{dist}(p, q_i) \leq \text{dist}(p', q_i) \) for any \( q_i \in Q \) and \( \text{dist}(p, q_i) < \text{dist}(p', q_i) \) 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
Note that making the location set $L$ in our FDL query be $Q$ as described above does not make the FDL query equivalent to the SSQ. The FDL query is not a skyline query characterized by multi-criteria optimization; the SSQ is still a skyline query in a transformed $n$-dimensional space, in which each dimension indicates the distance from a spatial point to a query point. The FDL query involves spatial distances in “ranking” and quality attributes in dominance definition; the basic form of SSQ does not involve any non-spatial attributes and define dominance based on distances to multiple query points. These major differences make the SSQ unsuitable for our FDL query. The extended version of SSQ [20], which takes into account non-spatial quality attributes, is still a skyline operator.

Huang et al. [14] 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 known line in the Euclidean space. Within the similar setting, Zheng et al. [27] addressed how to search for the result valid scope for such a query without a known moving pattern. Those works construct multiple dimensions of interest and then issue skyline queries on those dimensions to solve their specific problems. In contrast, our FDL query is not a skylining problem in a transformed space.

### 3 Algorithms for FDL Queries

Table 1 lists the notations used throughout the paper. Without the loss of generality, we assume that smaller values are preferred in quality attribute comparisons.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L$</td>
<td>a set of spatial locations (candidates)</td>
</tr>
<tr>
<td>$P$</td>
<td>a set of spatial objects (competitors)</td>
</tr>
<tr>
<td>$R_L$</td>
<td>an R-tree indexing the set $L$</td>
</tr>
<tr>
<td>$R_P$</td>
<td>an R-tree indexing the set $P$</td>
</tr>
<tr>
<td>$p.\psi$</td>
<td>$c$-dimensional vector (qualities of object $p$)</td>
</tr>
<tr>
<td>$\Psi$</td>
<td>$c$-dimensional vector (design competence)</td>
</tr>
<tr>
<td>$p.\psi[i]$ or $\Psi[i]$</td>
<td>$i$-th value of the vector</td>
</tr>
<tr>
<td>$\Psi \prec \Psi'$</td>
<td>$\Psi$ dominates $\Psi'$ (w.r.t. qualities)</td>
</tr>
<tr>
<td>$e$</td>
<td>a (spatially) minimum bounding rectangle</td>
</tr>
<tr>
<td>$\text{dist}(p,q)$</td>
<td>distance between two points/locations $p$ and $q$</td>
</tr>
<tr>
<td>$\text{mindist}(e,e')$</td>
<td>minimum distance between $e$ and $e'$</td>
</tr>
<tr>
<td>$\text{maxdist}(e,e')$</td>
<td>maximum distance between $e$ and $e'$</td>
</tr>
<tr>
<td>$\text{ndd}(s,\Psi, P)$</td>
<td>nearest dominator distance of location $s$ over $P$</td>
</tr>
<tr>
<td>$\Phi(e)$</td>
<td>bit value for the entry $e$ in a bitmap $\Psi$</td>
</tr>
</tbody>
</table>

#### 3.1 Baseline Algorithms

##### 3.1.1 Naive Iterative Incremental NN Search

We first present the naive iterative incremental NN search (NII) for processing the FDL query. NII takes as input (i) a 2-dimensional R-tree $R_P$ on the spatial object set $P$, (ii) a set of locations $L$, (iii) a $c$-dimensional design competence $\Psi$. It examines each location $s$ of set $L$ and computes the nearest dominator (ND) of $s$ by performing the incremental nearest neighbor search [12] of $s$ on the tree $R_P$. During the iterative search, the largest $\text{ndd}$ is maintained together with the corresponding location. After all locations are examined, the maintained location is returned as the query result.

#### Example of NII. In this example, the competitors’ set $P$ of objects contain $p_1, p_2, \ldots, p_6$, whose qualities are shown in Figure 3a, and their locations in the Euclidean space are depicted in Figure 3b. The R-tree $R_P$ (indexing $P$) has two root entries $e_1$ and $e_2$, pointing to a leaf node with $p_1, p_2, p_3$, and a leaf node with $p_4, p_5, p_6$ respectively.

Let the design competence $\Psi$ be $(3, 3)$. We need to find the nearest dominator of the location $s$ in Figure 3b. The NII algorithm retrieves the first NN object $p_1$ of $s$, whose quality vector $(4, 9)$ cannot dominate $\Psi$. Similarly, the next three NN objects $p_2, p_3, p_4$ also cannot dominate $\Psi$. After that, the next NN object $p_5$ has the quality vector $(2, 3)$, which dominates $\Psi$. Thus, the nearest dominator of $s$ is $p_5$. Observe that both leaf nodes are accessed.

#### 3.1.2 Enhanced Iterative Incremental NN Search

As shown above, the NII algorithm accesses many unnecessary objects (of $P$) that cannot dominate $\Psi$, before it finds the nearest dominator of the location $s$. Motivated by this observation, we propose to include additional information into non-leaf entries of the tree $R_P$, for reducing the number of accesses to those unqualified spatial objects (and R-tree nodes).

We propose an R-tree variant called $\psi$-augmented R-tree, for indexing the competitors’ object set $P$. Like the traditional R-tree, this tree also groups the nodes according to their spatial extent. The only difference is
that, in our ψ-augmented R-tree, each non-leaf entry \( e \) stores an additional \( q \)-dimensional vector \( e.\psi \). The \( i \)-th value of the vector is recursively defined as:

\[
e.\psi[i] = \min\{e'.\psi[i] | e' \text{ is in the child node of } e\}
\]  (2)

In words, the value \( e.\psi[i] \) indicates the minimum value of objects in its subtree, for the \( i \)-th quality attribute. The following lemma represents a pruning rule for pruning a non-leaf entry whose subtree cannot contain any object that dominates the design competence \( \Psi \).

**Lemma 1.** Let \( R_p \) be the ψ-augmented R-tree of the object set \( P \). Let \( e \) be a non-leaf entry of \( R_p \). If \( e.\psi \neq \Psi \), \( e' \)'s subtree cannot contain any object \( p \) dominating \( \Psi \).

Proof of this lemma is straightforward according to dominance transitivity. With this lemma, we extend the NII algorithm into the enhanced iterative incremental NN search algorithm (EII), as shown in Algorithm 1.

**Algorithm 1 EII**

(Augmented R-tree \( R_p \) of \( P \), Set \( L \), Competence \( \Psi \))

```plaintext
1: \( fb \) := null; \( ndd \) := 0
2: for each location \( s \in L \) do
3: \( s.ND := \) null; initialize a min-heap \( H \)
4: for each entry \( e' \) in \( R_p \)'s root do
5: if \( e'.\psi \prec \Psi \) then
6: enheap \((H, \langle e', mindist(s, e') \rangle)\)
7: while \( H \) is not empty and \( s.ND \) is null do
8: \( e := \) deheap \((H)\)
9: if \( e \) is a non-leaf entry then
10: for each child entry \( e' \) of \( e \) do
11: if \( e'.\psi \prec \Psi \) then
12: enheap \((H, \langle e', mindist(s, e') \rangle)\)
13: else
14: \( s.ND := e \)
15: if \( \text{dist}(s, s.ND) > \text{ndd} \) then
16: \( \text{ndd} := \text{dist}(s, s.ND); fb := s \)
17: return \( fb \)
```

Basically, the EII algorithm applies the incremental NN search [12] in Lines 3–16. The notation \( \text{mindist}(s, e) \) represents the minimum Euclidean distance between the location \( s \) and an R-tree entry \( e \) [19]. A min-heap \( H \) is used to organize its entries \( \langle e, \text{mindist}(s, e) \rangle \) to be visited in ascending order of the distance \( \text{mindist}(s, e) \).

The difference from [12] is that, in Lines 5–6 and 11–12, an entry \( e \) is enheaped only if \( e.\psi \prec \Psi \). This way, unnecessary entries are pruned and the number of node accesses is dramatically reduced.

**Example of EII.** The structure of the ψ-augmented R-tree (in Figure 3c) is the same as that of the traditional R-tree (in Figure 3b), except that each non-leaf entry \( e \) in the ψ-augmented R-tree stores its quality vector \( e.\psi \). For instance, the entry \( e_1 \) points to a leaf node containing \( p_1, p_2, p_3 \), whose quality vectors appear in Figure 3a. Thus, the entry \( e_1 \) stores \( e_1.\psi = (\min\{4, 6, 8\}, \min\{9, 7, 5\}) = (4, 5) \). Similarly, the entry \( e_2 \) stores \( e_2.\psi = (2, 1) \).

Applying Algorithm 1 to Figure 3c. The root entry \( e_1 \) is pruned as its quality vector \( e_1.\psi \) cannot dominate \( \Psi \). Next, the root entry \( e_2 \) is enheaped into \( H \) because its quality vector \( e_2.\psi \) dominates \( \Psi \). The entry \( e_2 \) is then deheaped from \( H \) and then its child entries \( p_3 \) are enheaped. Eventually, the object \( p_3 \) is deheaped and it is taken as the nearest dominator of \( s \).

**3.1.3 Access Locality Optimization**

Both the NII and EII algorithms are able to exploit an available memory buffer for reducing the number of accesses. To improve the access locality of the algorithms (and thus the buffer’s hit rate), it is beneficial to process the locations of \( L \) by the Hilbert curve ordering [7, 17].

Therefore, we sort the locations in \( L \) by the Hilbert curve ordering, and then process them accordingly in both NII and EII algorithms. In Section 5, we will study the effect of the Hilbert curve ordering.

**3.2 Best-First Search Algorithm**

Observe that the algorithms discussed above do not require any spatial index on the location set \( L \). In this section, we index the location set \( L \) by an R-tree \( R_L \), and exploit the grouping of these locations for accelerating the computation of the query result.

We first design a technique for computing the upper bound of nearest dominator (ND) distance of any location \( s \) in a group \( e_L \) (i.e., a rectangle). Then, we discuss how to prioritize the traversal on tree \( R_L \) based on such upper bound ND distance of tree entries.

**Deriving upper bound ND distance for a group.** Let \( \text{maxdist}(e, e') \) denote the maximum Euclidean distance between two rectangles \( e \) and \( e' \) [19]. Given the design competence \( \Psi \) and a minimum bounding rectangle \( e_L \) from \( R_L \), our question is how to derive the upper bound of the distance \( \text{ndd}(s, \Psi, P) \), for any possible location \( s \) in \( e_L \). The following lemma shows that the distance \( \text{ndd}(s, \Psi, P) \) is upper bounded by \( \text{maxdist}(e_L, p^*) \), when there is an object \( p^* \in P \) satisfying \( p^.*\psi \prec \Psi \).

**Lemma 2.** Let \( p^* \in P \) be an object such that \( p^.*\psi \prec \Psi \), where \( \Psi \) is the design competence. For any location \( s \) in a minimum bounding rectangle \( e_L \), it holds that \( \text{maxdist}(e_L, p^*) \geq \text{ndd}(s, \Psi, P) \).

**Proof:** Since \( p^.*\psi \prec \Psi \), the nearest dominator distance of \( s \) on \( P \) must be less than or equal to \( \text{dist}(s, p^*) \), i.e., we have \( \text{dist}(s, p^*) \geq \text{ndd}(s, \Psi, P) \). As \( s \) falls in the minimum bounding rectangle \( e_L \), we obtain \( \text{maxdist}(e_L, p^*) \geq \text{dist}(s, p^*) \). Combining both inequalities, we get \( \text{maxdist}(e_L, p^*) \geq \text{ndd}(s, \Psi, P) \).

Obviously, the above upper bound ND distance can be tightened by finding an object \( p^* \in P \) that satisfies \( p^.*\psi \prec \Psi \) and at the same time its location stays close to \( e_L \) in the Euclidean space.

Based on the above observation and Lemma 2, we develop Algorithm 2 for computing the above upper bound ND distance. It takes the following as input: the spatial object set \( P \)'s augmented R-tree \( R_p \), a minimum bounding rectangle \( e_L \) (of a group of locations in \( L \)), and the design competence \( \Psi \). The algorithm employs a min-heap to access all entries in \( R_p \) in ascending order of \( \text{mindist} \) between them and the rectangle \( e_L \). In addition,
Lemma 1 is applied to rule out irrelevant entries in $R_P$ (Lines 3–4 and 9–10). When a spatial object $e$ is met (Lines 11–12), it returns the distance $maxdist(e_L, e)$ as the upper bound ND distance for all locations in $e_L$.

The above algorithm regards the input $e_L$ as a rectangle. In case $e_L$ is a location (i.e., degenerated rectangle), the returned distance $maxdist(e_L, e)$ (at Line 12) degenerates to $dist(e, e)$. This means that the algorithm indeed computes the exact ND distance if $e_L$ is a location.

Algorithm 2 $score$(Augmented R-tree $R_P$ of $P$, Minimum bounding rectangle $e_L$ of a group of locations in $L$, Competence $\Psi$)

1: initialize a min-heap $H$
2: for each entry $e'$ in $R_P$’s root do
3:   if $e' \prec \Psi$ then
4:     enheap($H, \langle e', mindist(e_L, e') \rangle$)
5: while $H$ is not empty do
6:   $e :=$ deheap($H$)
7:   if $e$ is a non-leaf entry of $e$ do
8:     for each child $e'$ of $e$ do
9:       if $e' \prec \Psi$ then
10:      enheap($H, \langle e', mindist(e_L, e') \rangle$)
11:   else
12:  return $maxdist(e_L, e)$

Example of upper bound ND distance derivation. We then illustrate the above technique for computing the upper bound ND distance of $s$ in Figure 3d. Suppose that the design competence is $\Psi = (3, 3)$. The root entry $e_1$ is pruned as $e_1.\psi$ cannot dominate $\Psi$. Next, the root entry $e_2$ is enheaped into $H$ because $e_2.\psi$ dominates $\Psi$. After deheaping the entry $e_2$ from $H$, its child entries ($p_5$) that dominate $\Psi$ are enheaped into $H$. Eventually, the object $p_5$ gets deheaped from $H$ and the distance $maxdist(e_2, p_5)$ is returned as the upper bound ND distance of $e$ (shown as dotted line in Figure 3d).

Best-first search algorithm. We proceed to develop a best-first search algorithm that traverses the location set $L$’s R-tree $R_L$ in descending order of the upper bound ND distances of tree entries. The best-first search algorithm (BFS) is shown in Algorithm 3.

It uses a max-heap $H$ (Line 1) to organize R-tree entries encountered in the search, prioritizing those entries with higher scores which indicate farther dominated locations (Lines 3 and 8). Due to the property of the max-heap $H$ and the upper bounding property of $score(R_P, e', \Psi)$, the first location deheaped from the max-heap has the largest nearest dominator distance among all locations. In addition, $score(R_P, e, \Psi)$ refers to the exact ND distance of $e$ when $e$ is a location. Therefore, such a location $e$ is returned as the farthest dominated location (Line 10).

3.3 Spatial Join Based Algorithm

In both the EII and BFS algorithms, the $\psi$-augmented R-tree is used to index the spatial object set $P$. The extra field $\psi$ stored in a non-leaf entry is used to prune the search space; however, it does not fully prevent unnecessary accesses of tree nodes. Suppose that the design competence is $\Psi = (7, 6)$. In Figure 3c (and Figure 3d), despite the fact that the quality vector $e_1.\psi = (4, 5)$ of $e_1$ dominates $\Psi$, it does not guarantee that its subtree contains an object $p_i$ such that $p_i.\psi \prec \Psi$. In this example, all objects $p_1, p_2, p_3$ located within $e_1$ cannot dominate $\Psi$.

In following, we first use an R-tree $R_P$ to index the set $P$ and discuss how to compute a bitmap based on $P$. Then, we present a spatial join based algorithm that exploits the above bitmap to reduce the number of node accesses effectively.

Dominance bitmap. We proceed to give the definition of the dominance bitmap $\Phi$ as follows.

Definition 3: (Dominance Bitmap) Given the design competence $\Psi$ and an R-tree $R_P$ (for indexing the object set $P$), the dominance bitmap $\Phi$ stores a bit value $\Phi(e)$ as follows, for each non-leaf entry $e$ in $R_P$: The bit $\Phi(e)$ is set to 1 if the subtree of $e$ contains at least an object $p$ such that $p.\psi \prec \Psi$; otherwise, $\Phi(e)$ is set to 0.

At query time, we compute the dominance bitmap $\Phi$ by performing a complete depth-first traversal on the R-tree $R_P$. The bitmap occupies $f \cdot NL$ bits, where the average tree node fanout is $f$, and the tree contains $NL$ non-leaf nodes. The size of $\Phi$ is small and it fits into main memory. E.g., for an R-tree (with average fanout $f = 50$) containing 1 million objects, the bitmap occupies only 20000 bits, i.e., 2500 bytes.

We illustrate how to derive $\Phi$, by performing depth-first search on the R-tree in Figure 3b. Suppose that the design competence is $\Psi = (7, 6)$. After accessing the child node of $e_1$, none of its objects dominate $\Psi$ so we set $\Phi(e_1) = 0$. We then access the child node of $e_2$, find out that (at least) $p_5.\psi$ dominates $\Psi$ so we set $\Phi(e_2) = 1$.

Pruning rules using dominance bitmap. In the following, we discuss how the dominance bitmap can be used to prune unqualified entries from the trees $R_P$ and $R_L$.

Let $e_L$ be an entry of the tree $R_L$. Let $e_L.PL$ be the set of entries $e_P$ of the tree $R_P$ (visited so far) such that, for any location $s \in e_L.PL$ there exists an entry $e_P \in e_L.PL$ whose subtree contains the nearest dominator of $s$.

First of all, we apply the following pruning rule on $e_L.PL$ as any entry $e_i$ satisfying $\Phi(e_i) = 0$ cannot contain objects that dominate the design competence $\Psi$.

Pruning Rule 1: (Bitmap-based Pruning) Each non-leaf entry $e_i$ from $e_L.PL$ is pruned if $\Phi(e_i) = 0$. Each leaf entry $e_i$ from $e_L.PL$ is pruned if $e_i.\psi \not\prec \Psi$.

Algorithm 3 BFS(Augmented R-tree $R_P$ of $P$, R-tree $R_L$ of $L$, Competence $\Psi$)

1: initialize a max-heap $H$
2: for each entry $e'$ in $R_L$’s root do
3:   enheap($H, \langle e', score(R_P, e', \Psi) \rangle$)
4: while $H$ is not empty do
5:   $e :=$ deheap($H$)
6:   if $e$ is a non-leaf entry then
7:     for each child $e'$ of $e$ do
8:       enheap($H, \langle e', score(R_P, e', \Psi) \rangle$)
9:   else
10:  return $e$
After applying Pruning Rule 1, the subtree of each entry $e_i \in e_L.PL$ must contain at least an object $p^*$ that dominates $\Psi$. We then have the following lemma.

**Lemma 3:** Let $e_i$ be a minimum bounding rectangle such that it contains an object $p^* \in P$ satisfying $p^*.x < \Psi$, where $\Psi$ is the design competence. For any location $s$ in a rectangle $e_L$, it holds that $\text{maxdist}(e_L, e_i) \geq ndd(s, \Psi, P)$.

**Proof:** We get $\text{maxdist}(e_L, p^*) \geq ndd(s, \Psi, P)$ by Lemma 2. We have $\text{maxdist}(e_L, e_i) \geq \text{maxdist}(e_L, p^*)$ since $e_i$ contains $p^*$. Thus we get: $\text{maxdist}(e_L, e_i) \geq ndd(s, \Psi, P)$.

By Lemma 3, the distance $\text{maxdist}(e_L, e_i)$ provides an upper bound of the distance $\text{ndd}(s, \Psi, P)$ for any location $s \in e_L$. Suppose that Pruning Rule 1 has been applied on $e_L.PL$. By taking the minimum of these upper bounds over all $e_i$’s in $e_L.PL$, we define the upper bound $\text{ndd}$ distance of the rectangle $e_L$ as follows:

$$UBndd(e_L) = \min_{e_i \in e_L.PL} \text{maxdist}(e_L, e_i) \tag{3}$$

The upper bound distance $UBndd(e_L)$ of the entry $e_L$ enables us to apply: (i) Pruning Rule 2, a local pruning technique for discarding entries $e_i \in e_L.PL$ that do not contain nearest dominator of any location in $e_L$, and (ii) Pruning Rule 3, a global pruning technique for removing an entry $e_L$ (and its set $e_L.PL$) that does not lead to better (higher) ND distances.

**Pruning Rule 2: (Local Pruning)** Each entry $e_i$ from $e_L.PL$ is pruned if $\text{mindist}(e_L, e_i) > UBndd(e_L)$.

**Pruning Rule 3: (Global Pruning)** Let $s' \in L$ be a location, and $p' \in P$ be its nearest dominator object with respect to the design competence $\Psi$. An entry $e_L$ (together with its set $e_L.PL$) is pruned if $\text{dist}(s', p') > UBndd(e_L)$.

These two rules are easy to prove with the definition of $UBndd(e_L)$. Figure 4a illustrates the derivation of the upper bound distance $UBndd(e_L)$ of the entry $e_L$ (from the tree $R_L$). In this example, the set $e_L.PL$ contains three entries $e_{p1}, e_{p2}, e_{p3}$, all of them have the bit value 1 in the dominance bitmap $\Psi$. We then compute $UBndd(e_L) = \min(\text{maxdist}(e_L, e_{p1}), \text{maxdist}(e_L, e_{p2}), \text{maxdist}(e_L, e_{p3})) = \{0.5, 0.6, 0.9\} = 0.5$. The local pruning rule is applied to remove $e_{p3}$ from $e_L.PL$, since $\text{mindist}(e_L, e_{p3}) = 0.65 > UBndd(e_L) = 0.5$. Note that $e_{p2}$ cannot be pruned, because for a location $s \in e_L, ND(s, \Psi, P)$ may come from $e_{p2}$ though $\text{maxdist}(e_L, e_{p2}) > \text{maxdist}(e_L, e_{p1})$.

In Figure 4b, we have found a location $s'$ with its ND distance as 0.53, which is greater than $UBndd(e_L) = 0.5$. Thus, the global pruning rule is used to discard the entry $e_L$ and its set $e_L.PL$ from searching.

**Spatial join based algorithm.** With the above pruning rules, we are ready to present our spatial join based algorithm (SJB). Its pseudo-code is shown in Algorithm 4. Each heap entry is of the form $(e_L, e_L.PL, UBndd(e_L))$, where $e_L$ is an entry in the tree $R_L, e_L.PL$ is the set of potential tree entries of $R_P$ that could be joined with $e_L$ to produce better results. Entries in $e_L.PL$ are sorted in ascending order of $\text{mindist}(e_L, e_j)$ where $e_j \in e_L.PL$. A max-heap $H$ is employed for organizing its heap entries to be visited in descending order of upper bound distances $UBndd(e_L)$ of its entries.

**Algorithm 4 SJB**(R-tree $R_P$ of $P$, R-tree $R_L$ of $L$, Competence $\Psi$)

1. initialize a max-heap $H$
2. create the dominance bitmap $\Phi$ for $R_P$ w.r.t. $\Psi$
3. $e_{root} := R_L.root; e_{root}.PL := \{R_P.root\}$
4. enqueue($H, (e_{root}, e_{root}.PL, 0)$)
5. while $H$ is not empty do
6. $(e_L, e_L.PL) := \text{deheap}(H)$
7. if $e_L$ is a leaf entry then
8. if $e_L.PL$ contains only leaf entries then
9. return $e_L$
10. else go to Line 20
12. else
13. if $e_L.PL.first$ is an object or
14. an $R_P$ entry is expanded in previous iteration then
15. for each entry $e_i$ in $e_L do$
16. $e_i.PL := e_L.PL$
17. apply local pruning (Rule 2) on $e_i.PL$
18. enqueue($H, (e_i, e_i.PL, UBndd(e_i))$
19. else
20. $e_P := e_L.PL.first$
21. remove the first entry from $e_L.PL$
22. for each entry $e_i$ in $e_P do$
23. if $e_i$ is a dominating object or $\Phi(e_i) = 1$ then
24. add $e_i$ to $e_L.PL$
25. apply local pruning (Rule 2) on $e_L.PL$
26. enqueue($H, (e_L, e_L.PL, UBndd(e_L))$
26. During the joining, $R_P$ entries and $R_L$ entries are basically expanded in an alternative way (Lines 13-26). An $R_L$ entry $e_L$ is expanded as follows. For each subentry $e_j$ in $e_L$, all joining entries from $e_L$ are copied (Line 16), local pruning (Pruning Rule 2) is then applied on $e_i.PL$, and finally $e_i$ is inserted into max-heap $H$ with its upper bound distance $UBndd(e_i)$ (Lines 17-18).

When an $R_P$ entry is to be expanded, the first one in the current $e_L$’s $PL$ list is picked as $e_P$ and removed from the list (Lines 20-21). Each subentry $e_j$ in this $e_P$ is added to $e_i.PL$, if $e_j$ itself is an object that dominates $\Psi$ or it is an entry that contains dominating object(s) in its subtree (i.e., Pruning Rule 1). After all subentries of $e_P$ have been processed, the entry $e_L$ together with its $e_L.PL$ set and upper bound distance $UBndd(e_L)$, is reinserted into max-heap $H$ (Lines 25-26). The algorithm
returns when a location and a spatial object is to be joined, according to Pruning Rule 3 (Lines 8–9).

Our SJB algorithm differs from the spatial join and R-tree join (RJ) algorithm [6]. They consider only the spatial relationship between the objects in $S$ and $P$, but not their dominance relationship on quality attributes of objects in our FDL query. Our SJB algorithm also differs from RJ algorithm in that SJB does not employ recursion. Instead, SJB algorithm joins $S$ and $P$ incrementally, integrating our specific dominance check efficiently in each increment step.

### 3.4 Analysis of Pruning Effectiveness

In this section, we analyze the pruning effectiveness of: (i) a $\psi$-augmented tree (used in EII and BFS) and (ii) an R-tree with a dominance bitmap $\Phi$ (used in SJB).

Suppose that there are $N$ objects in the object set $P$. For the sake of analysis, we assume that the quality vectors of these objects are uniformly distributed in the domain $[0,1]^c$, where $c$ is the number of quality attributes/dimensions.

Observe that the $\psi$-augmented tree achieves a grouping of nodes similar to the R-tree, since both of them group the objects into leaf nodes solely based on their spatial locations. Thus, the following analysis focuses on only the quality dimensions but not spatial locations.

Let the average fanout of both R-trees be $f^j$. Observe that an entry at tree level $j$ contains $f^j$ objects in its subtree. Let $e.QR$ be a hyper-rectangle in the space $[0,1]^c$, defined as the minimum bounding rectangle of quality vectors (of objects) in $e$’s subtree. Note that $e.QR$ is only used as a convenient notation in our analysis, it is not explicitly stored in our trees. According to Theodoridis et al. [23], the side length $\lambda_j$ of $e.QR$ equals to:

$$\lambda_j = \left(\frac{f^j}{N}\right)^{\frac{1}{c}} \quad (4)$$

We proceed to examine the example of Figure 5, with $c = 2$ quality attributes/dimensions. Specifically, we consider three design competencies $\Psi_1$, $\Psi_2$, $\Psi_3$ (for different queries) and check whether a non-leaf entry $e$ in the tree can be pruned. We first study the case of $\psi$-augmented tree and then investigate the case of an R-tree with dominance bitmap.

Figure 5a depicts the augmented quality vector $e.\psi$ of the entry $e$, in the $c$-dimensional space defined by quality dimensions. In case the design competence falls into the gray region (e.g., $\Psi_1$), we guarantee that $e.\psi$ cannot dominate $\Psi_1$ and thus the entry $e$ is pruned (in the EII/BFS algorithms). On the other hand, the child node of $e$ needs to be accessed when the design competence stays in the remaining region (e.g., $\Psi_2$, $\Psi_3$).

Figure 5b illustrates the bit value $\Phi(e)$ for the entry $e$ in the dominance bitmap $\Phi$. Recall from Section 3.3 that, the dominance bitmap is constructed by a complete depth-first traversal of the tree. Regarding the design competence $\Psi_1$, none of the points in $e.QR$ can dominate $\Psi_1$, so we have $\Phi(e) = 0$, meaning that the entry $e$ will get pruned (in the SJB algorithm). For the case of $\Psi_2$, any point in $e.QR$ dominates $\Psi_2$, we obtain $\Phi(e) = 1$ and the entry $e$ cannot be pruned. Notice that the dominance bitmap has the same pruning effectiveness as the $\psi$-augmented tree, for the design competencies $\Psi_1$ and $\Psi_2$.

The real advantage of using the dominance bitmap is that, even if a design competence (e.g., $\Psi_3$) resides in $e.QR$, it is still possible to obtain $\Phi(e) = 0$ (and prune the entry $e$), for the scenario that the subtree of $e$ indeed contains no objects with quality vectors dominating $\Psi_3$ (i.e., empty space in gray area). Our next question is to derive the probability of such an event.

The following analysis considers the case that the design competence $\Psi$ falls into $e.QR$. For simplicity, we assume that the $i$-th coordinate $\Psi[i]$ of $\Psi$) deviates from the $i$-th dimension lower bound value of $e.QR$, by an offset value $r$. Suppose that the entry $e$ is at the $j$-th tree level. As illustrated in Figure 5b, our objective is to find out the probability that the subtree of $e$ contains no objects in the gray region. Observe that the ratio of the gray region’s area to $e.QR$’s area is given by: $(r/\lambda_j)^c$.

Since the subtree of $e$ contains $f^j$ objects, the probability that the bit value $\Phi(e)$ equals to 0 is given by:

$$Pr(\Phi(e) = 0) = \left(1 - \left(\frac{r}{\lambda_j}\right)^c\right)^{f^j} = \left(1 - \frac{Nr^c}{f^j}\right)^{f^j} \quad (5)$$

Interestingly, the probability rises rapidly when $c$ increases. Thus, the dominance bitmap offers better pruning power at higher dimensionality.

Continuing with the example of Figure 5b, we have $r = \lambda_j/4$ for the design competence $\Psi_3$. Substituting the value $j = 1$ and $f = 10$ into the equation, we obtain $Pr(\Phi(e) = 0) = 0.52$ for the case of $\Psi_3$.

### 3.5 Handling Undominated $\Psi$

So far we have implicitly assumed that the design competence $\Psi$ is dominated by some object(s) in $P$. As a matter of fact, it is possible that an input $\Psi$ is too advantaged to be dominated. For such an input, SJB...
algorithm is able to stop with reasonable overhead to create the dominance bitmap only, whereas none of the other algorithms stop early.

To avoid unnecessary high overhead, we adjust these algorithms slightly to make them able to handle undominated $\Psi$s efficiently. For the iterative incremental NN search based algorithms (NII and EII), after processing the first location we check if the result $fdl$ is null. If positive, which means the input $\Psi$ is not dominated, we return without processing any remaining location. As a result, the overhead for NII is to retrieve all spatial objects via the incremental NN search; the overhead for EII is much lower because of the pruning effectiveness in the augmented R-tree $R_P$.

For BFS algorithm, the score algorithm (Algorithm 2) returns nothing if the input design competence $\Psi$ is not dominated. We instead let it return a value that cannot be the distance $\maxdist(e_l, e)$, e.g., $-\infty$. Accordingly, we in BFS check each score value returned. If it is $-\infty$, we discard the corresponding entry $e'$ and exit immediately. This way, the overhead for BFS to handle an undominated $\Psi$ is merely calling score algorithm for the first entry in $R_L$ root. The effect will be seen in Section 5.2.

4 Technical Generalizations

4.1 Generalization of Distance Metrics

So far we have employed the Euclidean distance as the distance metric in the FDL query definition. However, other distance metrics, e.g., the road network distance [21] and Manhattan distance, are also of importance in many scenarios. Therefore, we in this section generalize the FDL query definition and algorithms to support generic distance metrics.

Let $dist_g(o_1, o_2)$ denote the generic distance between two spatial objects or locations. Given a location $s$, its quality vector $\Psi$, and a set of spatial objects $P$, the definitions of ND and ndd in Section 1.1 are generalized as follows.

**Definition 4:** (Generalized Nearest Dominator, Generalized Nearest Dominator Distance)

$$ND_g(s, \Psi, P) = \arg\min_{o \in P, o \neq s} dist_g(s, o)$$

$$ndd_g(s, \Psi, P) = dist_g(s, ND_g(s, \Psi, P))$$

Accordingly, the FDL query is generalized as follows.

**Definition 5:** (Generalized Farthest Dominated Location Query) Given a set of (competitors’) spatial objects $P$, a set of (candidate) locations $L$, and a quality vector $\Psi$ as the design competence, the generalizable farthest dominated location query (FDL$_g$) returns from $L$ a location $s$ such that the generic distance $ndd_g(s, \Psi, P)$ is maximized, i.e., $\forall s' \in L$, $ndd_g(s, \Psi, P) \geq ndd_g(s', \Psi, P)$.

To process the FDL$_g$ query, our algorithms for Euclidean distance based FDL query are generalized as follows.

Generalizing NII is straightforward. Instead of using R-tree as the index, we use the M-tree [9] to index the spatial object set $P$ based on the generic distance metric. Accordingly, the incremental nearest neighbor search on R-tree is replace by its counterpart on M-tree [12]. According to the definition of M-tree, all objects in a subtree $T$ are within the covering radius $T.cr$ from the routing object $T.ro$. Given a location $s$ and a M-tree subtree $T$, the lower bound distance between them, i.e., $mindist_g(s, T)$ is $dist_g(s, T.ro) - T.cr$.

To generalize the EII algorithm, we augment the M-tree on $P$ as we have done in Section 3.1.1, adding a $c$-dimensional vector $e.\Psi$ in each node entry of M-tree $M_P$. Then, by replacing $d_l$ with $dist_g$ and $mindist_g$ with $mindist_g$ defined above, we generalize EII (Algorithm 1) to the generic distance metric.

To generalize the BFS algorithm, we also use the augmented M-tree $M_P$ instead of the augmented R-tree $R_P$ on $P$. In addition, we use the M-tree $M_L$ instead of the R-tree $R_L$ on $L$. Accordingly, we generalize the score algorithm (Algorithm 2) by using the generic $mindist_g$ and $\maxdist$. Specifically, the generalized $\maxdist$ between a location $s$ and a M-tree subtree $T$ is $\maxdist_g(s, T) = dist_g(s, T.ro) + T.cr$. The generalized $mindist_g$ between two M-tree subtrees $T_1$ and $T_2$ is $mindist_g(T_1, T_2) = dist_g(T_1.ro, T_2.ro) - T_1.cr - T_2.cr$.

To generalize the SJB algorithm, we use the M-tree $M_P$ on $P$ without augments. The dominance bitmap $\Psi$ is created by a depth-first traversal on $M_P$. All the pruning rules still work, with generic $mindist_g$ and $\maxdist$ replacing their Euclidean counterparts. In particular, the upper bound defined in Equation 3 should use $\maxdist_g$ for two M-tree subtrees. Specifically, $\maxdist_g(T_1, T_2) = dist_g(T_1.ro, T_2.ro) + T_1.cr + T_2.cr$.

Figure 6 illustrates the generalized distance metrics.

![Fig. 6. Generalized $mindist_g$ and $\maxdist_g$](image)

4.2 Nearest Dominated Location Queries

The FDL queries are intended to maximize the nearest dominator distance (ndd) and find the location with the largest ndd. Sometimes, it is of interest to know the location with the smallest ndd. Such locations indicate the least competitive option (e.g., in business planning), or the most endangered option (e.g., in wild animal protection). Identifying them also makes sense as they are the worst cases that decision makers want to avoid.

Therefore, we define the Nearest Dominated Location Query as follows.

**Definition 6:** (Nearest Dominated Location Query) Given a set of (competitors’) spatial objects $P$, a set of (candidate) locations $L$, and a quality vector $\Psi$ as
the design competence, the nearest dominated location query (NDL) returns from \( L \) a location \( s \) such that the Euclidean distance \( \text{ndd}(s, \Psi, P) \) is minimized, i.e., \( \forall s' \in L, \text{ndd}(s, \Psi, P) \leq \text{ndd}(s', \Psi, P) \)

We proceed to adapt our FDL algorithms to process NDL queries. It is apparent that both our NII and EII algorithms can be easily modified to process NDL queries. To modify both of them, the minimum \( \text{ndd} \) and the corresponding location in \( L \) are maintained in the iteration on all locations in \( L \). At the end of the iteration, the maintained location is returned as the nearest dominated location.

To adapt BFS, the challenge is to figure out the appropriate score function for a location entry \( e_L \). It should be the lower bound (instead of the upper bound) of \( \text{ndd} \) for any location \( s \in e_L \). Particularly, score\((R_P, e_L, \Psi) \) \( = \min\{\text{mindist}(e_L, e_P) \mid e_P \in R_P \land e_P \psi \prec \Psi \} \). The modified score function, to be called by the adapted BFS, is presented in Algorithm 5. For a leaf entry \( e_L \), the score actually is its \( \text{ndd} \) (lines 9–11). Otherwise, the score calculation stops at level 1 of R-tree \( R_P \) on \( P \) (lines 6–8). This way, we are able to derive a tight lower bound without incurring too many node accesses. Also, the max-heap used in BFS should be replaced by a min-heap to give priority to smaller \( \text{ndd} \).

Algorithm 5 NDL_score(Augmented R-tree \( R_P \) of \( P \), Minimum bounding rectangle \( e_L \) of a group of locations in \( L \), Competence \( \Psi \))

```
1: for each entry \( e' \) in \( R_P \)'s root do 
2:    if \( e' \psi \prec \Psi \) then 
3:        enheap\((H, \langle e', \text{mindist}(e_L, e') \rangle)\)) 
4: while \( H \) is not empty do 
5:    \( e := \text{deheap}(H) \) 
6:    if \( e_L \) is a non-leaf entry then 
7:        if \( e \) is a level-1 entry then 
8:            return mindist\((e_L, e)\) 
9:        else 
10:            if \( e \) is a leaf entry then 
11:                return mindist\((e_L, e)\) 
12:            if \( e \) is a non-leaf entry then 
13:                for each child \( e' \) of \( e \) do 
14:                    if \( e', \psi \prec \Psi \) then 
15:                        enheap\((H, \langle e', \text{mindist}(e_L, e') \rangle)\)) 
```

When adapting the SJB algorithm to NDL queries, two points are noteworthy. First, Pruning Rule 1 still works as the dominance bitmap still tells which \( P \) R-tree nodes are unqualified. Second, Pruning Rule 2 also works as it discards \( P \) entries that do not contain nearest dominator of any location in a location entry. To return the nearest dominated location, we need a lower bound \( \text{LBndd}(e_L) \) in addition to the upper bound (Equation 3). The lower bound should be defined as

\[
\text{LBndd}(e_L) = \min_{e_i \in e_L, P_L} \text{mindist}(e_L, e_i) \quad (8)
\]

Accordingly, the max-heap in SJB is replaced by a min-heap that prioritizes \( L \) entries according to the aforementioned lower bound instead of the upper bound.

Following the same line of reasoning as in Section 4.1, NDL queries and the relevant processing algorithms can be generalized to generic distance metric. Due to the space limitation, we omit the details in this paper.

5 Experimental Study

In this section, we conduct intensive experiments to evaluate and compare those farthest dominated location query processing algorithms: NII, EII, their Hilbert curve variants (NII-Hil and EII-Hil respectively), BFS and SJB. All algorithms were implemented in Java and were run on a Windows XP PC with a 2.8GHz Intel Pentium D CPU and 1GB RAM. We used real and synthetic datasets for both object set \( P \) and location set \( L \). For each dataset, its spatial coordinates were normalized to Euclidean space \([0, 10,000] \times [0, 10,000]\); while all its quality attributes were normalized to space \([0, 1] \)^c, where \( c \) is the number of quality attributes. Each coordinate and attribute value is of 8-byte double type.

We set the page size to 4K bytes for both data and R-tree indexes. Each pointer in R-trees uses 4 bytes. The page fanout in the location R-tree \( (R_L) \) is 113; that in other R-trees \( (R_P) \) varies from 35 to 78. We used an LRU memory buffer whose default size was set to 0.5% of the sum of data sizes. We measure both the IO cost (i.e., number of page faults) and total response time of each query, and report the average measure obtained from 10 instances of \( \Psi \) randomly drawn from the \( c \)-dimensional space. If not explicitly stated, then each generated \( \Psi \) is dominated by some existing object. For SJB algorithm, the construction cost of the dominance bitmap (e.g., a traversal of the tree \( R_P \)) is included in the measurement.

5.1 FDL Query Performance on Real Datasets

In this part, we used two real datasets from AllStays.com [1] that maintains datasets of hotels, resorts, campgrounds, etc. around the world. We chose the dataset of hotels in US and cleaned it up as follows. We used different quality attribute combinations from 30,918 hotel records were obtained for AllStays.com that maintains datasets of hotels, resorts, campgrounds, etc. around the world. We chose the dataset of hotels in US and cleaned it up as follows. We removed all records without longitude and latitude, and gave up quality attributes with too many null values. For all quality attributes remained, any null value was replaced by a value randomly picked from its attribute domain. As a result, 30,918 hotel records were obtained with the schema \( \text{longitude, latitude, review, stars, price} \).

We then normalized all 30,918 hotel records as described above, and call the normalized dataset USH. Value conversion was carried out on a quality attribute if necessary, e.g., a higher stars value was converted to a lower value in the normalized range \([0, 1]\). This way, lower values are preferable to higher ones. After that, two thirds (20,612) records are randomly picked as the basis to generate the \( P \) datasets. Specifically, we used different quality attribute combinations from USH and got four variants of USH dataset, as shown in Table 2. Each USH variant was used as the \( P \) set in our experiments. Whereas all locations of the remaining one third (10,306) hotel records form the \( L \) dataset. The results are reported in Figure 7.
The node access of each algorithm is shown in Figure 7a. It is not surprising that NII algorithm performs worst because it carries out a naive search without any pruning. Then NII-Hil, the NII version processing all locations in \( L \) according to the Hilbert curve ordering (see Section 3.1.3), performs slightly better than NII. The Hilbert ordering of all locations helps to increase the buffer hit ratio, however, the naive nature of NII provides little space for improvement. Because of the \( \psi \)-augmentation in the R-tree \( R_P \) on object set \( P \), EII algorithm is able to prune considerable number of unqualified nodes and thus incurring significantly fewer node accesses.

![Fig. 7. Performance on real datasets](image)

Table: Use of USH dataset in experiments

<table>
<thead>
<tr>
<th>USH Variant</th>
<th>Quality Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>USH-rs</td>
<td>review, stars</td>
</tr>
<tr>
<td>USH-rp</td>
<td>review, price</td>
</tr>
<tr>
<td>USH-sp</td>
<td>stars, price</td>
</tr>
<tr>
<td>USH-rsp</td>
<td>review, stars, price</td>
</tr>
</tbody>
</table>

5.2 FDL Query Performance on Synthetic Datasets

We generated synthetic datasets and evaluated all farthest dominated location algorithms on them. The cardinality of synthetic \( P \) sets changes from 100K to 1000K. Its quality attribute dimensionality changes from 2 to 5. For all quality attributes in each \( P \) set, we generated values following both independent (IN) distribution and anti-correlated (AC) distribution in the same way introduced in previous work [5]. For each \( P \) set, the cardinality of the corresponding \( L \) set used in the experiment is a percent of that of \( P \). The percent changes from 10% up to 60%. For both \( P \) sets and \( L \) sets, all locations in each set are generated randomly in the normalized Euclidean space \([0, 10,000]\) \( \times [0, 10,000] \). The parameters of all synthetic datasets are listed in Table 3. Those default settings are shown in bold fonts.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object set card., (</td>
<td>P</td>
</tr>
<tr>
<td>Quality attri. dimen., (c)</td>
<td>2, 3, 4, 5</td>
</tr>
<tr>
<td>Quality attri. distri.</td>
<td>Indep.(IN), Anti-corre. (AC)</td>
</tr>
<tr>
<td>Location set card., (</td>
<td>L</td>
</tr>
</tbody>
</table>

Table 3: Parameters of synthetic datasets

5.2.1 Pruning Effectiveness

In this part, we study the pruning effectiveness of augmented R-tree and dominance bitmap with respect to design competence \( \Psi \). We accordingly used augmented R-trees and basic R-trees on 100K spatial object sets. We changed the quality attribute dimensionality \( c \) from 2 to 5. Each design competence \( \Psi \) is a \( c \)-dimensional vector of the form \((v, \ldots, v)\), where \( v \in \{0.1, 0.3, 0.5, 0.7, 0.9\} \).

For each basic R-tree, we create the dominance bitmap with respect to the given design competence \( \Psi \). For each R-tree (either augmented or basic), we perform a depth-first traversal to access each node that may contain some object(s) dominating the \( \Psi \). We measure the percent of R-tree nodes accessed in the depth-first traversal.

According to the results reported in Figure 8, dominance bitmaps always exhibit better pruning effectiveness than augmented R-trees. Compared to an augmented R-tree, a counterpart dominance bitmap has no false positive because of its bit definition. Therefore, it helps rule out considerable number of R-tree nodes which has a dominating \( \psi \) in the counterpart augmented R-tree. Dominance bitmaps obtain higher pruning effectiveness when the dimensionality is higher. In an attribute space of higher dimensionality, a design competence is expected to be dominated by fewer objects when the object cardinality is fixed. A dominated bitmap is able to adapt to this and set fewer bits to 1, which renders more R-tree nodes to be pruned.

Note \( \Psi \) values 0.1 and 0.3 are not included in the experiments on AC attributes, as in shown Figure 8b. This is because their corresponding \( \psi \)'s are not dominated by any spatial object with AC attributes.
We validated the analysis of pruning effectiveness in Section 3.4, by comparing the probability obtained from Equation 5, i.e. the estimated R-tree node pruning ratio, with the actual R-tree node pruning ratio in the experiments. We validated on IN quality attributes which the analysis assumes. According to Figure 8(a), we fixed $v$ to 0.1. We also focused on R-tree leaf nodes because upper levels contain insufficient nodes for validation. According to the validation results shown in Figure 9, Equation 5 tends to make underestimates compared to the actual pruning effects.

### 5.2.2 Effect of Design Competence $\Psi$

We also investigated the effect of design competence $\Psi$, using the default synthetic dataset settings: 100K $P$ with 2-dimensional quality attributes, and 20% of $|P|$ location set $L$. Each design competence $\Psi$ is of the form $(v, v)$, where $v \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$. The results on node access are reported in Figure 10.

On IN quality attributes, each design competence $\Psi$ is dominated. Referring to Figure 10a, NII-Hil incurs higher overhead if $\Psi$ is closer to $(0, 0)$. Such a $\Psi$ is dominated by fewer objects. As a result, NII-Hil has to search a larger spatial range in object R-tree $R_P$, and thus accessing more R-tree nodes, before finding the nearest dominator for each individual location. Whereas the effect is different to other algorithms. Such high node accesses do not apply to EII-Hil because it is able to prune unpromising nodes when $\Psi$ is closer to $(0, 0)$. A $\Psi$ closer to $(0, 0)$ causes $\psi$ in an $R_P$ entry unlikely to be dominating. Therefore fewer entries are accessed by the score algorithm in BFS. A $\Psi$ closer to $(0, 0)$ also renders fewer bits to be set in the dominance bitmap, which causes fewer entries to be accessed by SJB.

The results on AC quality attributes are shown in Figure 10b. Here the first two $\Psi$s, $(0.1, 0.1)$ and $(0.3, 0.3)$, are dominated by no objects in $P$. In these cases, each algorithm stops early without processing all locations, as discussed in Section 3.5. Nevertheless, NII-Hil is still the worse because it has no pruning at all; SJB is still the best for it only needs to create the dominance bitmap. For other cases, the results are similar to those on IN quality attributes.

### 5.2.3 Effect of Buffer Size

We also investigated into the effect of buffer size with all other parameters set to default. We varied the buffer size from 0.25% to 2% of the sum of data sizes. The relevant results are shown in Figure 11.

As expected, larger buffer size helps reduce node access. For datasets with IN attributes, larger buffer size benefits NII-Hil and EII-Hil most, as shown in Figure 11a. Whereas for datasets with AC attributes, SJB gains most from a larger buffer, according to Figure 11b. A given design competence $\Psi$ is more likely to be dominated by points of IN distribution. This renders a candidate location more likely to be dominated by nearby neighbors. As a result, when all locations are accessed according to the proximity preserving Hilbert curve, more nodes are likely to be found in the buffer.

On the other hand, the total query response time is not too sensitive to the buffer size value. This is because the in-memory processing of each algorithm is not affected by the buffer size.

### 5.2.4 Experiments on Scalability

In this part, we study the scalability of FDL query processing algorithms with respect to data cardinality and dimensionality. In each experiment, we report the query response time.

**Effect of $P$ Cardinality.** We first fixed the quality attribute dimensionality to 2, $L$ cardinality to 20% of $P$ cardinality, and varied the $P$ cardinality to observe its impact on the performance of all algorithms. Relevant results are reported in Figure 12a and Figure 13a.

Figure 12a plots the results on synthetic datasets with independent (IN) attributes. Our SJB algorithm outperforms all other three ones: NII-Hil, EII-Hil and BFS. The performance difference between SJB and others is mainly attributed to the pruning effectiveness discrepancy between the dominance bitmap and the augmented R-tree. The slight higher overhead of BFS compared to EII-Hil is due to two factors. First, the pruning effectiveness of augmented R-tree used by BFS is low for low dimensionality, as we see in Section 5.2.1. This makes the score algorithm tend to access more nodes for each input $R_L$ entry $e_L$. Second, BFS involves two heaps (in BFS itself...
and the score algorithm) which consume more time in their specific operations.

Figure 11a plots the results of response time on synthetic datasets with anti-correlated (AC) attributes. The superiority of SJB to others is more significant compared to datasets with IN attributes. A random design competence \( \Psi \) has lower chance to be dominated by a set of anti-correlated points than by a set of independent points. This means that all algorithms but SJB access more nodes before the ND is located. Whereas, SJB employs the dominance bitmap, which saves considerable node access in query processing. Note BFS gains advantage over NII-Hil and EII-Hil, which is attributed to Lemma 2 favored by anti-correlated quality attributes.

**Effect of L Cardinality.** We then fixed the \( P \) cardinality to 100K, the attribute dimensionality to 2, and varied \( L \) cardinality to observe its impact on the performance. Relevant results are reported in Figure 12b and Figure 13b.

Referring to Figures 12b, the increase of \( L \) cardinality causes longer query response time for each algorithm on independent quality attributes. NII-Hil still performs worst, whereas SJB is still the best.

Referring to Figures 13b, NII-Hil performs much worse than others when \( L \) cardinality increases on AC quality attributes, as it is not equipped with any pruning mechanism. SJB still performs best and scales very slowly as \( L \) cardinality increases. Whereas BFS degrades moderately and outperforms both NII-Hil and EII-Hil, because the augmented R-tree is more powerful in terms of pruning on anti-correlated quality attributes.

**Effect of Quality Dimensionality.** Finally, we fixed \( |P| \) to 100K, \( |L| \) to 20% of \( |P| \), and varied the quality attribute dimensionality from 2 to 5. Relevant results are reported in Figure 12c and Figure 13c.

Both NII-Hil and EII-Hil degrade seriously when dimensionality goes up, especially on anti-correlated (AC) quality attributes. In contrast, BFS degrades more slightly, especially on anti-correlated (AC) quality attributes. As the dimensionality becomes higher, the chance for the design competence \( \Psi \) to dominate a given design competence \( \Psi \) is lower. This, together with the effect of AC distribution, renders the augmented R-tree \( R_P \) higher pruning capability. When working on a higher dimensionality following AC distribution, the score algorithm called by BFS benefits more from augmented R-tree \( R_P \), because it utilizes the pruning mechanism more often for location groups from R-tree \( R_L \).

It is noteworthy that SJB is still the best algorithm on both attribute distributions; it even improves as dimensionality increases. Given a design competence \( \Psi \) and a fixed number of points, \( \Psi \) tends to be dominated by fewer points if the dimensionality increases. This renders fewer nodes set in the resulting dominance bitmap, which subsequently saves processing time in the spatial
joining employed by SJB algorithm.

5.3 Performance of Generalizations

We proceed to evaluate our generalization proposals presented in Section 4. We used the real San Joaquin County road network [4] with 18,263 vertices and 23,797 edges. We randomly picked 20% of the vertices as the location set $L$, and attached 2 to 5 quality attributes (both IN and AC distributions) to the remaining vertices to obtain the object sets $P_s$. The network distance metric was used in all FDL queries.

The node access costs are reported in Figure 14. The SJB algorithm outperforms the alternatives by 2–3 orders of magnitude. This indicates that the pruning techniques employed by SJB works effectively and efficiently on generic distance based M-trees.

![Fig. 14. Network distance based FDL queries](image)

(a) IN attributes  (b) AC attributes

The node access costs of network distance based NDL queries are reported in Figure 15. The SJB algorithm is still the best. Referring to Figure 15a, objects in $P$ with IN attributes have similar likelihood to be a dominator, which renders the lower bound (Equation 8) used in SJB less effective in pruning candidate M-tree entries. Referring to Figure 15b, $P$ objects with AC attributes differ markedly in terms of likelihood to be a dominator, which makes more room for the lower bound used in SJB to prune candidate M-tree entries. This trend is intensified as the dimensionality increases.

![Fig. 15. Network distance based NDL queries](image)

(a) IN attributes  (b) AC attributes

6 Conclusion

In this paper, we propose a novel complex type of query: farthest dominated location (FDL) query. Given a set of (competitors’) spatial objects $P$ with both spatial locations and non-spatial attributes, a set of (candidate) locations $L$, and a design competence vector $\Psi$ (for $L$), a FDL query retrieves the location $s \in L$ such that the distance to its nearest dominating object in $P$ is maximized. Although FDL queries are suitable for various spatial decision making applications, they are not solved by any of the existing techniques.

We develop several efficient R-tree based algorithms for processing FDL queries, which offer users a range of selections in terms of different indexes available on the data. We also generalize our proposals to support the generic distance metric and other interesting query types. We conduct an extensive experimental study with various settings on both real and synthetic datasets. The results disclose the performance of our proposals, and identify our spatial joint based algorithm (SJB) as the most efficient and scalable query processing algorithm.

References

Hua Lu received the BSc and MSc degrees from Peking University, China, in 1998 and 2001, respectively and the PhD degree in computer science from National University of Singapore in 2007. He is currently an assistant professor in the Department of Computer Science, Aalborg University, Denmark. His research interests include skyline queries, spatio-temporal databases, geographic information systems, and mobile computing. He is a member of the IEEE.

Man Lung Yiu received the bachelors degree in computer engineering and the PhD degree in computer science from the University of Hong Kong in 2002 and 2006, respectively. Prior to his current post, he worked at Aalborg University for three years starting in the Fall of 2006. He is now an assistant professor in the Department of Computing, Hong Kong Polytechnic University. His research focuses on the management of complex data, in particular query processing topics on spatiotemporal data and multidimen-