General Spatial Skyline Operator

†Qianlu Lin †Ying Zhang †Wenjie Zhang §Aiping Li
{qlin, yingz, zhangw}@cse.unsw.edu.au, apli1974@gmail.com

†The University of New South Wales Australia,
§National University of Defense Technology, China

Abstract. With the emergence of location-aware mobile device technologies, communication technologies and GPS systems, various location-aware queries have attracted great attentions in the database literature. In many user recommendation systems, the spatial preference query is used to suggest the objects based on their spatial proximity to the facilities. In this paper, we study the problem of *general spatial skyline* which can provide a minimal set of candidates that contain optimal solutions for any monotonic distance based spatial preference query. An efficient algorithm is proposed to significantly reduce the number of non-promising objects in the computation. The paper also covers a comprehensive performance study of the proposed techniques based on both real and synthetic data.

1 Introduction

With the development of mobile device technologies, communication technologies and GPS systems in recent years, there has been an increasing number of location based service systems specialized in providing interesting results through location based queries which retrieve the desirable candidate objects for users based on the spatial proximity of the objects and facilities. For instance, as shown in Figure 1(a), there are a set of apartments, bus stations and supermarkets in the map, and a user wants to rent an apartment which is close to both a bus station and a supermarket. In Figure 1(b), each apartment is mapped to a point in a 2-dimensional space where the distances to the nearest bus station and supermarket are the coordinate values of an apartment. As shown in Figure 1 the apartment a_4 derives its coordinates from the distance to its closest bus station (b_1) and supermarket(s_1). Clearly, the smaller value is preferred. As there is no apartment with both shortest supermarket-distance and bus station-distance in the example, the user needs to make a trade-off. Suppose the user has a preference function against the distances of an apartment regarding its closest bus station and supermarket, the system can return the apartment with best score regarding the preference function. If the preference function is in the form of $f(o) = 4 \times o.d_1 + o.d_2$ where $o.d_1$ and $o.d_2$ represent the distances of o to the closest supermarket and bus station respectively, then a_4 is the best choice. The answer becomes a_3 if we have $f(o) = o.d_1 + 4 \times o.d_2$. This is the distance based spatial preference query[1], and the problem is studied in [10, 17,

[1] See Section 2.2 for the formal definition

12]. However, in many applications users cannot find an appropriate preference function. Therefore, it is desirable to provide a minimal candidate set for users so that they can make personal trade-offs without missing any potential optimal solution.

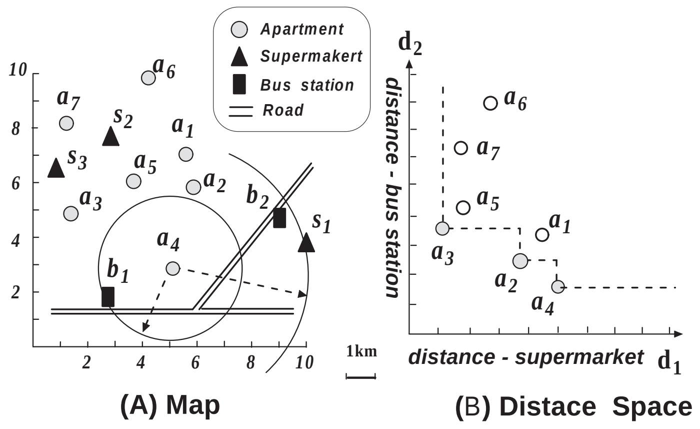

Fig. 1. Motivating Example

Motivated by the above example, in this paper we propose the *general spatial skyline* (*GSSKY*) operator. Given a set $\mathcal{O}$ of objects and a set $\mathcal{F}$ of facilities with m types, an object o can be mapped to a point $\tilde{o}$ in m-dimensional space, named distance space, where the coordinate value on i-th dimension is the distance of o to its nearest facility with type i. We say an object o_1 *spatially dominates* another object o_2 if $\tilde{o_1}$ *dominates* $\tilde{o_2}$ in the mapped distance space. Note that the *dominance* relationship in distance space is the same as the traditional skyline problem [1]; that is, we say $\tilde{o_1}$ *dominates* $\tilde{o_2}$ if $\tilde{o_1}$ is not larger than $\tilde{o_2}$ on any dimension $i \in [1, m]$, and $\tilde{o_1}$ is smaller than $\tilde{o_2}$ on at least one dimension. Then the objects which are not *spatially dominated* by any other object are *general spatial skyline* objects. As shown in Section 2.2, the *general spatial skyline* objects can provide a minimal set of candidates that contain optimal solutions for any monotonic distance based spatial preference query. Moreover, we show theoretically and experimentally that the number of *GSSKY* objects is usually much smaller than that of the objects.

Note that although there are some existing works [14, 15] which study the problem of *spatial skyline*, they cannot provide a minimal set for the distance based spatial preference queries studied above due to the essential differences between the two problems. Please see Section 6 for detailed discussion.

Challenge. A straightforward solution for the *GSSKY* query is to compute distance values of all objects and then apply the traditional skyline algorithm. This is not efficient because the distance computation (i.e., retrieving the distance of an object to the closest facility regarding a particular type) is expensive and we have to compute the distance values for all objects. In this paper, we propose a novel *GSSKY* computation algorithm which aims to reduce the amount of distance computations by pruning non-promising objects. Our contributions can be summarized as follows.

- The *general spatial skyline* query is formally defined, that provides a minimal set of candidates which contain optimal solutions for any monotonic distance based spatial preference query.
- An efficient algorithm is proposed to compute the *general spatial skyline*.
- Comprehensive experiments demonstrate the efficiency of our techniques.

The remainder of the paper is organized as follows. We formally define the problem and discuss related techniques in Section 2. Section 3 presents the all nearest neighbor based algorithm. Section 4 proposes our efficient *GSSKY* algorithm. Results of a comprehensive performance study are presented in Section 5. Section 6 presents the related work. Finally, Section 7 concludes the paper.

2 Preliminary

In Section 2.1, we formally define the problem of general spatial skyline computation . In Section 2.2, we show that the general spatial skyline can provide a minimal set of candidates that contain optimal solutions for any monotonic spatial preference function. We introduce the incremental nearest neighbor algorithm in Section 2.4. Table 1 below summarizes the mathematical notations frequently used.

Notation	**Definition**
o ($\mathcal{O}$)	object (a set of objects)
f ($\mathcal{F}$)	facility (a set of facilities)
m	the number of facility types in $\mathcal{F}$
$\mathcal{F}_i$	all facilities in $\mathcal{F}$ with type i
$o.d_i$	the distance between o and its closest facility with type i
$o \lhd \mathcal{F}$	o is *fully hit* by $\mathcal{F}$
r_i	the maximal hit distance seen so far regarding facilities with type i
$o_1 \prec_{\mathcal{F}} o_2$	o_1 *spatially dominates* o_2 regarding $\mathcal{F}$
$GSSKY(\mathcal{O}, \mathcal{F})$	the general spatial skyline of $\mathcal{O}$ regarding the facilities $\mathcal{F}$

Table 1. The summary of notations.

2.1 Problem Definition

A point x referred in this paper, by default, is in a d-dimensional numerical space. Let $\delta(x, y)$ denote the Euclidian distance between two points x and y [2]. In the paper, $\mathcal{F}$ represents a set of facilities and $\mathcal{F}_i$ denotes all facilities in $\mathcal{F}$ with type i. A facility f is a point in the space with a particular facility type.

An object o is a point in a d-dimensional numerical space. The distance of o to $\mathcal{F}_i$, denoted by $o.d_i$, is the distance between o and its closest facility with type i, i.e., $o.d_i = \min$ ($\delta(o, f)$ for any $f \in \mathcal{F}_i$). As shown in Figure 1, given a

[2] We focus on Euclidian distance in the paper. Nevertheless, our techniques can be easily extended to other L_p norm distances.

set $\mathcal{F}$ of facilities with m categories, an object o can be mapped to a point in m-dimensional space. We define the *spatial dominance* relationship as follows.

Definition 1 (Spatial Dominance). *Given two objects o_1, o_2 and a set $\mathcal{F}$ of facilities, We say object o_1 spatially dominates another object o_2 regarding $\mathcal{F}$, denoted by $o_1 \prec_{\mathcal{F}} o_2$, if and only if $o_1.d_j \leq o_2.d_j$ for* ***any*** *facility type j, and there is a facility type i such that $o_1.d_i < o_2.d_i$.*

Example 1. In Figure 1, we have $a_2 \prec_{\mathcal{F}} a_1$, $a_3 \prec_{\mathcal{F}} a_5$, and $a_4 \not\prec_{\mathcal{F}} a_5$.

Based on the *spatial dominance* relation, we come up with the definition of *general spatial skyline* as follows.

Definition 2 (General Spatial Skyline). *Given a set $\mathcal{O}$ of objects and a set $\mathcal{F}$ of facilities, the general spatial skyline of $\mathcal{O}$ regarding $\mathcal{F}$, denoted by $GSSKY(\mathcal{O}, \mathcal{F})$, are objects which are not spatially dominated by any other objects regarding $\mathcal{F}$.*

Example 2. In Figure 1, we have $GSSKY(\mathcal{O}, \mathcal{F}) = \{a_2, a_3, a_4\}$.

Problem Statement.
In this paper we investigate the problem of efficiently computing general spatial skyline for a set of objects with respective to multiple types of facilities.

2.2 Minimal Set Property

Given a set $\mathcal{O}$ of objects and a set $\mathcal{F}$ of facilities with m types, the score of an object o regarding $\mathcal{F}$, denoted by o_s, is derived based on its closest facilities. Following is a formal definition of the distance based spatial preference query.

$$o_s = p(\ o.d_1,\ \ldots, o.d_m) \tag{1}$$

Recall that $o.d_i$ denotes the distance between o and its closest facility with type i. For presentation simplicity, we use "spatial preference function" to abbreviate "distance based spatial preference function" in the paper whenever there is no ambiguity. The following theorem indicates that the $GSSKY$ provides the minimal set for all increasing spatial preference functions.

Theorem 1. *Let $\mathcal{P}$ denote the family of all increasing spatial preference functions regarding $\mathcal{F}$, for any $p \in \mathcal{P}$ the object with best score is in $GSSKY(\mathcal{O}, \mathcal{F})$. For any object o in $GSSKY(o, \mathcal{F})$, there exists a spatial preference function $p \in \mathcal{P}$ such that o has the best score regarding p.*

Proof. For any object $o_2 \notin GSSKY(\mathcal{O}, \mathcal{F})$, there is an object o_1 such that $o_1 \in GSSKY(\mathcal{O}, \mathcal{F})$ and $o_1 \prec_{\mathcal{F}} o_2$ according to the definition of $GSSKY$. We have $o_1.d_j \leq o_2.d_j$ for any $j \in [1, m]$ and there exists $i \in [1, m]$ such that $o_1.d_i < o_2.d_i$. According to the monotonic property of the functions, we have $p(o_1) < p(o_2)$ for any increasing spatial preference function p. With similar rationale, there is an increasing spatial preference function p for each object $o \in GSSKY(\mathcal{O}, \mathcal{F})$ such that $p(o)$ has lowest score among all objects. Therefore, the theorem holds. ■

2.3 Size Estimation

Based on [5], we have the following theorem which estimates the size of *GSSKY* objects using an independence assumption.

Theorem 2. *Suppose the locations of the facilities and the objects are independent to each other, then the expected number of GSSKY object is* $O(\frac{(\ln(n))^{m-1}}{(m-1)!})$ *where* n *is the number of objects in* $\mathcal{O}$*.*

2.4 Incremental Nearest Neighbor Technique

As our general spatial skyline algorithm proposed in Section 4 is based on the incremental nearest neighbors(INN) computation, we introduce the INN technique [7] in this subsection. Unlike the k nearest neighbor query where k is known beforehand, the INN algorithm will incrementally output the next closest neighbor , i.e., the $(l+1)$-th nearest neighbor where l is the number of neighbors seen so far, on user's demand.

Suppose the objects are organized using an R-tree. A priority queue $\mathcal{Q}$ is used to maintain a set of R-tree entries (intermediate entries and data entries) where the key of an entry is its minimal distance to the query point. The root of the R-tree is pushed into $\mathcal{Q}$ at the beginning of the algorithm. For each incremental nearest neighbor request, the algorithm outputs the data entry in $\mathcal{Q}$ with smallest key value. Note that we say an object is in $\mathcal{Q}$ if its corresponding data entry or any of its ancestor entries is in $\mathcal{Q}$. Specifically, if the entry with smallest key value is a data entry which is associated with an object o, o is output and popped from $\mathcal{Q}$. Otherwise, the intermediate entry (i.e., index or leaf node) is popped and expanded, and all its child entries are pushed into $\mathcal{Q}$. The procedure is repeated until the entry on top of $\mathcal{Q}$ is a data entry. The algorithm can therefore be used to incrementally determine the next nearest neighbor. [7] has shown the efficiency of the INN algorithm theoretically and experimentally.

3 All Neareast Neighbor(ANN) based GSSKY Algorithm

Since the *GSSKY* problem is exactly the same as the traditional skyline problem if all objects are mapped to the distance space $\mathcal{D}$, a straightforward solution for the *GSSKY* computation is to first compute the distances for all objects regarding $\mathcal{F}$, and then apply the existing skyline algorithm. As the computation of the distance values of the objects regarding facilities with type i can be achieved by all nearest neighbor (ANN) queries against $\mathcal{O}$ and $\mathcal{F}_i$, in this subsection, we apply the state-of-the-art ANN technique [2] to compute the *GSSKY*. Algorithm 1 outlines the ANN based general spatial skyline computation. Note that all existing non-index skyline techniques can be applied in Line 3 once the distance values of all objects are available. As shown in the empirical study, the dominant cost of Algorithm 1 is the distance computation.

Algorithm 1: ***ANN based GSSKY*** $(\mathcal{O}, \mathcal{F})$

```
Input  : $\mathcal{O}$ : the objects,
         $\mathcal{F}$ : the facilities
Output : $\mathcal{S}$ : GSSKY( $\mathcal{O}$, $\mathcal{F}$)
1 for each facility type $i$ in $[1..m]$ do
2     Compute the $o.d_i$ for each object $o \in \mathcal{O}$ by applying ANN [2] algorithm
      against $\mathcal{O}$ and $\mathcal{F}_i$ ;
3 $\mathcal{S} \leftarrow$ compute skyline on the distances of the objects;
4 return $\mathcal{S}$
```

4 Efficient *GSSKY* Algorithm

4.1 Motivation

As shown in the empirical study, the dominant cost of the *GSSKY* computation comes from the calculation of the distance values for the objects. Consequently, even if we apply the state-of-the-art technique to compute the distance values for all objects, the ANN based *GSSKY* algorithm is still inefficient in terms of both I/O and CPU costs. Motivated by this, in this section we aim to reduce the number of distance computations during the *GSSKY* query process.

In the paper, we may compute the distance values of the objects in two ways:

Object Oriented Search.
For each object o, we compute the $o.d_i$ by applying the nearest neighbor(NN) algorithm [13] where o is the query point. For instance, as shown in Figure 1 $o_4.d_1$ and $o_4.d_2$ can be derived by issuing two NN queries against $\mathcal{F}_1 = \{s_1, s_2, s_3\}$ and $\mathcal{F}_2 = \{b_1, b_2\}$ respectively, where o_4 is the query point. Particularly, the all nearest neighbor (ANN) algorithm [2] can also be considered as an object oriented method in which the object distances are computed in a batch fashion.

The advantage of the object oriented search is that, for a given object or a set of objects, we can directly derive the distances of the objects. However, as there is no a priori knowledge about the distance values of the *unvisited* objects, like Algorithm 1 in Section 3, we have to compute distance values for all objects to ensure the correctness of *GSSKY* computations.

Facility Oriented Search.
Instead of computing distance values for each individual object, we can derive them by applying incremental nearest neighbor(INN) algorithm against facilities simultaneously where the query point is a facility. As shown in Figure 2, for each facility $f \in \mathcal{F}$, we maintain a radius f_r and we say an object o has been *hit* by f if $\delta(f, o) \leq f_r$. The distance between o and f is called the hit distance of o regarding f. Similarly, we say an object o is *fully hit* by $\mathcal{F}$, denoted by $o \lhd \mathcal{F}$, if o has been hit by **all types** of facilities; that is, for any $\mathcal{F}_i$, there exists a facility $f \in \mathcal{F}_i$ such that o is *hit* by f. For each facility type i, we maintain a global radius r_i which is the maximal hit distance seen so far regarding facilities with type i.

At each iteration, for each type i we find a facility f in $\mathcal{F}_i$ to invoke a new hit by expanding f_r such that the increment of r_i is minimized. Clearly, the global radius r_i is non-decreasing in the search. Due to the monotonic property

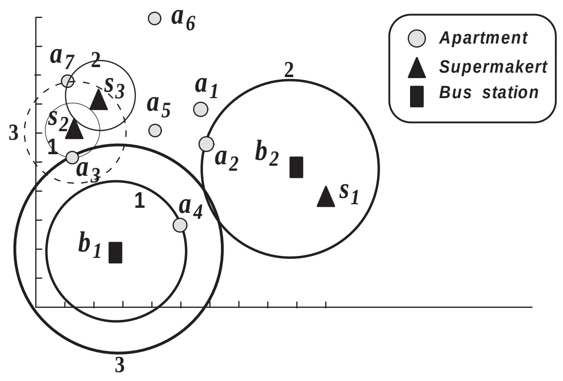

Fig. 2. Running Example

of r_i, we can safely set $o.d_i$ to the hit distance when it is *hit* for the first time by a facility with type i. Recall that an object may be *hit* multiple times by the facilities with the same type. Therefore, we say a *hit* is a *redundant hit* if the object has been *hit* by another facility with the same type.

Example 3. Figure 2 illustrates a snapshot of the facility oriented search in which we use a circle to record each hit of the objects. Specifically, circles with thin(bold) line represent the hits from bus stations (supermarkets) and the number of a circle indicates the accessing order. Moreover, the circle with solid (dashed) line represents a *non-redundant hit* (*redundant hit*). In Figure 2, $a_3.d_1$ and $a_4.d_2$ are derived in the first iteration. In the third iteration, the hit of a_7 regarding s_3 is a *redundant hit* because a_7 has been *hit* by s_2 in the second iteration.

Without loss of generality, in the paper we assume the hit distance is distinct for each facility type. Note that the duplication can be easily handled by visiting all objects with the same hit distance. Because of the monotonic property of the hit distance (i.e., r_i), the following lemma is immediate, which enables us to obtain the lower bound of the distance values for the *unvisited* objects.

Lemma 1. *In the facility oriented search, we have $o.d_i > r_i$ if an object o has not been hit by any facility with type i so far.*

Based on Lemma 1, the following theorem implies that we can safely prune some objects from the $GSSKY(\mathcal{O}, \mathcal{F})$ without any distance computation.

Theorem 3. *In the facility oriented search, suppose there exists an object o_1 which has been hit by **all types** of facilities, an object o_2 can be pruned from $GSSKY(\mathcal{O}, \mathcal{F})$ if o_2 has not been hit by **any** facility.*

Proof. We have $o_1.d_i \leq r_i$ for any $i \in [1, m]$ since o_1 has been *hit* by all types of facilities. On the other hand, we have $o_2.d_i > r_i$ for any object o_2 which has not been *hit* by any facility. It is immediate that $o_1 \prec_{\mathcal{F}} o_2$ and hence o_2 can be pruned from $GSSKY(\mathcal{O}, \mathcal{F})$. Therefore, the theorem holds. ∎

Example 4. In Figure 2, objects $\{a_1, a_5, a_6\}$ can be pruned from $GSSKY(\mathcal{O}, \mathcal{F})$ without any distance computation because none of them has been *hit* by any facility when a_3 is *fully hit*.

Another advantage of the facility oriented search is that, as shown in [7], the amortized cost for each hit distance computation in INN query is cheaper than that of a NN query because the INN algorithm can share the computation by continuously maintain the priority queue. This implies that if the proportion of the *redundant hits* is not significant, the facility oriented method is more efficient. Intuitively, the proportion of the *redundant hits* will increase with the global radius r_i regarding $\mathcal{F}_i$ because the larger the radius, the higher chance an object is *hit* by multiple facilities in $\mathcal{F}_i$. Another disadvantage of the the facility oriented search is that we need to maintain a priority queue for each facility and it is not space efficient when the number of facilities is very large.

Motivated by the advantages and disadvantages of the object oriented search and the facility oriented search, we propose an efficient $GSSKY$ algorithm which combine both methods in an effective way. The algorithm consists of three phases. In the first phase, we apply the facility oriented search to compute object distances and prune objects (i.e., remove non-skyline objects) based on lemma 1 and Theorem 3. This is feasible because the number of facilities is usually much smaller than that of objects in real applications. When we find that the computation of facility oriented search becomes less efficient due to the large amount of *redundant hits*, the algorithm goes to phase two, in which we compute the distances of the remaining objects based on the object oriented search (i.e., NN query). Finally, in phase three we apply the existing skyline algorithm to finalize the $GSSKY$ computation.

4.2 Algorithm

In the paper, we assume a set $\mathcal{O}$ of objects are organized using R-Tree, denoted by $R_{\mathcal{O}}$, and all facilities with type i are also organized using R-Tree $R_{\mathcal{F}_i}$. The Algorithm 2 illustrates the details of the efficient $GSSKY$ algorithm.

In Line 2-9, we apply the facility oriented search to compute the distances of the objects until there exists an object which has been *fully hit*. Particularly, a *local priority queue* is employed for each facility f for INN query, i.e., retrieve the next closest neighbor of f. For each facility type $i \in [1, m]$, we use a *global priority queue* $\mathcal{Q}_i$ to maintain the current closest neighbors (i.e., objects) of the facilities in $\mathcal{F}_i$. The elements *global priority queue* $\mathcal{Q}_i$ are prioritized using distances. In Line 4, the object o on the top of $\mathcal{Q}_i$ is popped and a INN query is issued by its associated facility to retrieve the next closest neighbors o_2. Then o_2 is pushed into $\mathcal{Q}_i$. Line 6 sets $o.d_i$ to the current hit distance (recorded by r_i) if it is a *non-redundant hit*. When the loop is terminated (Line 9), o is a $GSSKY$ object and kept in $\mathcal{S}$, and objects which have been *hit* at least once are kept in the candidate set $\mathcal{C}$. According to Theorem 3, all remaining objects can be pruned. For I/O efficiency, we keep the page ids of the nodes (i.e., intermediate entries) of the $R_{\mathcal{O}}$ visited so far. In the following facility oriented search (Line 10-21), we do not access a node of $R_{\mathcal{O}}$ if its page id is not recorded (i.e., all of its descendant data entries correspond to the pruned objects) and hence the I/O cost can be saved.

Algorithm 2: **Efficient *GSSKY* Algorithm** ($\mathcal{O}$, $\mathcal{F}$)

```
Input  : $\mathcal{O}$ : the objects,
         $\mathcal{F}$ : the facilities with $m$ types
Output : $\mathcal{S}$ : GSSKY( $\mathcal{O}$, $\mathcal{F}$)
1  $\mathcal{S} := \emptyset$; $\mathcal{C} := \emptyset$; $r_i := 0$ for each $\mathcal{F}_i$;
2  while true do
3      for each facility type $i$ in $[1..m]$ do
4          $o \leftarrow$ next object in facility oriented search regarding $\mathcal{F}_i$;
5          if the hit of $o$ is a non-redundant hit then
6              $o_{d_i} := r_i$; $\mathcal{C} := \mathcal{C} \cup o$;
7          if $o$ is fully hit by $\mathcal{F}$ then
8              $\mathcal{S} := o$; $\mathcal{C} := \mathcal{C} - o$;
9              Terminate the while loop;
10 while true do
11     for each facility type $i$ in $[1..m]$ do
12         $o \leftarrow$ next object in facility oriented search regarding $\mathcal{F}_i$;
13         if $o$ is a candidate object and the hit of $o$ is a non-redundant hit then
14             $o_{d_i} := r_i$;
15             if SkylineTest($\mathcal{S}$, $o$) then
16                 if $o$ is fully hit by $\mathcal{F}$ then
17                     $\mathcal{C} := \mathcal{C} - o$; $\mathcal{S} := \mathcal{S} + o$;
18             else
19                 $\mathcal{C} := \mathcal{C} - o$;
20         if #redundant hit is larger than #non-redundant hit then
21             Terminate the while loop;
22 for each object $o \in \mathcal{C}$ do
23     calculate $o_{d_i}$ by NN query if $o$ has not been hit regarding $\mathcal{F}_i$ ;
24 for each object $o \in \mathcal{C}$ accessed in non-decreasing order based on $\sum_{i=1}^{m} o_{d_i}$ do
25     if SkylineTest($\mathcal{S}$, $o$) then
26         $\mathcal{S} := \mathcal{S} \cup o$;
27 return $\mathcal{S}$
```

In Line 10-21, we continue the facility oriented search and try to identify the *GSSKY* objects and prune the non-promising ones. Particularly, if the object o output in Line 12 is a candidate object (i.e., $o \in \mathcal{C}$) and the hit is a *non-redundant hit*, Line 15 checks if there exits an object $s \in \mathcal{S}$ (i.e., *GSSKY* objects seen so far) such that $s \prec_{\mathcal{F}} o$. Note that if o has not been *hit* by any facility with type i, o_{d_i} is temporarily set to r_i in the test. We say an object o passes the skyline test (**SkylineTest**) if it is not *spatially dominated* by any object in $\mathcal{S}$. In the case o passes the test (Line 4.2-17), it is a *GSSKY* object **if** o has been *fully hit*. Otherwise, we cannot claim that o is a *GSSKY* object at this moment because the lower bound of the distance is employed in the skyline test. Line 19 eliminates the object from $\mathcal{C}$ if o fails the test. As discussed in Section 4.1, the facility oriented seasrch should be stopped when r_i becomes large due to the increased probability of *redundant hit*. However, it is impossible to find the optimal stop time without knowing the exact distributions of the following *redundant hits* and *non-redundant hits* are unknown. In the paper, we employ a simple but effective criteria. The number of *non-redundant hits* and *redundant hits* are counted, and Line 21 terminates the facility oriented search if there are more *redundant hits*.

Line 22-23 calculate the missing distances for the objects in the candidate set, where the object oriented search is employed. Recall that the missing of o_{d_i} value implies that the object o has not been *hit* by any facility with type i so far. The remaining part of the algorithm is similar to the SFS Algorithm [3]. Particularly, we sort all the candidate objects based on the sum of their distance values (i.e., $\sum_{i=1}^{m} o_{d_i}$) in *non-decreasing* order (Line 22), and Line 25 checks if an object is *spatially dominated* by the *GSSKY* objects ($\mathcal{S}$) seen so far. The objects passed the test are *GSSKY* objects (Line 26).

Correctness. For the correctness of the Algorithm 2, we need the following properties: (i) any object pruned at Line 15 and Line 25 are not *GSSKY* object, (ii) all objects *unvisited* in Algorithm 2 are not *GSSKY* objects, and (iii) the object in $\mathcal{S}$ cannot be dominated by any other object in $\mathcal{O}$. Below is a formal proof.

Proof. The correctness of the property (i) is immediate based on the definition of *GSSKY* if o has been *fully hit* at Line 15 and Line 25. If o_{d_i} is replaced by r_i at Line 15 (i.e., o has not been *hit* by any facility with type i) and o is dominated by an object $s \in \mathcal{S}$, we can claim that s *spatially dominates* o regarding $\mathcal{F}$ due to the monotonic property of r_i (Lemma 1). The correctness of property (ii) is immediate based on Theorem 3.

We prove the correctness of property (iii) by the contradiction. Suppose the object s is in $\mathcal{S}$ but s is *spatially dominated* by another object o. We can assume o is a *GSSKY* object because of the *dominance transitivity* property of *spatial dominance*, i.e., $o_1 \prec_{\mathcal{F}} o_2$ and $o_2 \prec_{\mathcal{F}} o_3$ implies $o_1 \prec_{\mathcal{F}} o_3$. If s is put in $\mathcal{S}$ at Line 4.2 or Line 17, o should be included in $\mathcal{S}$ before s. This is because $o \prec_{\mathcal{F}} s$ implies s is *fully hit* after o due to the monotonic property in the facility oriented search. This contradicts the proposition that s is the first object being *fully hit* (). Also, s should also fail in the test in and s will not be added into $\mathcal{S}$. We can come up with similar contradiction if s is put in $\mathcal{S}$ at Line 26 because we access objects based the sum of their distance values. ■

Performance Analysis. Upon each hit in Algorithm 2 (Line 2-21), an INN query is issued to retrieve the next closest neighbor of a facility $f \in \mathcal{F}_i$ where $i \in [1, m]$ and the global priority queue $\mathcal{Q}_i$ is updated. The cost is $C_{inn} + O(\log(n_f))$ where C_{inn} and n_f denote the average cost of a INN query and the average number of facilities for each type respectively. If it is a *non-redundant hit*, the skyline test is invoked which costs $O(|\mathcal{S}|)$ in the worst case where $|\mathcal{S}|$ is the size of $\mathcal{S}$. In Line 22-23, the cost is $O((m-1) \times |\mathcal{C}| \times C_{nn})$ in the worst case where C_{nn} is the average cost for NN query and $|\mathcal{C}|$ denotes the candidate set size. Recall that a candidate object will be *hit* at least once. The sorting cost in Line 24 is $O(|\mathcal{C}| \times \log(|\mathcal{C}|))$ and the cost of skyline computation in Line 24-26 is $|\mathcal{S}|^2$ in the worse case. In summary, let n_r and n_s denote the number of *redundant hits* and *non-redundant hits*, the time complexity of Algorithm 2 is $O((n_r+n_s)\times(C_{inn}+\log(n_f)) + n_s\times|\mathcal{S}| + (m-1)\times|\mathcal{C}|\times C_{nn} + |\mathcal{C}|\times\log(|\mathcal{C}|)+|\mathcal{S}|^2)$. Note that, in practice $|\mathcal{S}|$ and $|\mathcal{C}|$ are much smaller than the total number of objects, and hence the algorithm is quite efficient.

Following theorem estimate the number of objects accessed, i.e., objects have been *hit* at least once, in Algorithm 2 based on the uniform and independence assumption.

Theorem 4. *Suppose that (i) objects and facilities are uniformly distributed in the space* $[0,1]^2$*, (ii) there are* n_f *facilities for each type, and (iii) the locations are independence regarding different types of facilities. The expected number of objects accessed in Algorithm 2 is* $n(1-(1-\pi\bar{X}^2)^m)$ *where* n *is the number of objects. Particularly, we have* $\bar{X}$ *equals* $\int_{r=0}^{c}(1-F(r))'r\ d(r)$ *where* $c = \frac{1}{\sqrt{2}n_f}$*, and* $F(r) = (1-(n_f\pi r^2)^m)^n$.

Proof. Due to the space limitation, we give a brief proof. According to the uniform assumption and each type has the same number of facilities, we can assume $r_i = r_j$ in each iteration where $1 \le i, j \le m$. Therefore, we use r to denote the r_i for any $i \in [1, m]$. The probability that none of the objects is fully *hit* for given r, denoted by $F(r)$, is $(1-(n_f\pi r^2)^m)^n$ due to the uniform and independence assumption. Let X denote the distance r when the first object is fully *hit*, then its expected value $\bar{X}$ equals $\int_{r=0}^{c}(1-F(r))' \times r\ d(r)$ where $c = \frac{1}{\sqrt{2}n_f}$. Consequently, the expected number of objects accessed is $n(1-(1-\pi\bar{X}^2)^m)$. ■

5 Performance Evaluation

In this section, We present the results of a comprehensive performance study to evaluate the efficiency and scalability of the proposed techniques in the paper. The following algorithms were selected for evaluation.

ANN The all nearest neighbor based technique presented in Section 3. The SFS algorithm [3] is used in Algorithm 1 for skyline computation.

GSSKY The efficient *GSSKY* algorithm proposed in Section 4.

Both algorithms in this paper are implemented in standard C++ with STL library support and compiled with GNU GCC. Experiments are run on a PC with Intel Xeon 2.40GHz dual CPU and 4G memory running Debian Linux. The disk page size is fixed to 4096 bytes.

Real datasets. Two real spatial datasets, *CA* and *US*, are employed in the experiment[3]. *CA* consists of 104,217 locations of 44 different categories (e.g., church, lake and school). Each category corresponds to a facility type. The objects in *CA* are constructed as follows. We first normalize the space to $[0,1]^2$ and

[3] *CA* is available from http://www.cs.fsu.edu/ lifeifei/SpatialDataset.htm. *US* is available from http://www.geonames.usgs.gov/.

then for each facility we randomly create 5 objects within distance 0.005. Consequently the number of objects in *CA* dataset is 521, 085. Similarly, *US* dataset is obtained from the U.S. Geological Survey (USGS) and consists of 406, 709 locations with 40 types. The number of objects in *US* is 2, 033, 545.

Synthetic datasets. To study the scalability of the algorithms, we also create synthetic dataset, denoted by SYN, in the experiment. The objects and facilities are randomly generated in 2-dimensional space $[0, 1]^2$. Specifically, the number of objects varies from $500K$ to $5M$ with default value $1M$. There are 40 types of facilities and the number of facilities for each type varies from 500 to 10, 000 with default value 2, 000. SYN is the default dataset in the experiment.

Work load. The work load of each experiment consists of 200 $GSSKY$ queries and m types are randomly chosen in each query where m varies from 2 to 5 with default value 3. In the paper, the average processing time, which includes the CPU time and I/O latency, is used to measure the efficiency of the algorithms. We also record the average $GSSKY$ size and the average number of nodes loaded.

Table 2 lists parameters which may have an impact on our performance study. In our experiments, all parameters use default values unless otherwise specified.

Notation	**Definition (Default Values)**
m	the number of facility types (3)
n	the number of objects ($1M$)
n_f	the number of facilities for each type (2000)

Table 2. System Parameters

5.1 *GSSKY* SIZE

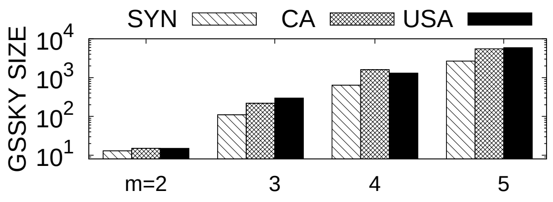

Fig. 3. Diff. datasets and m

In this subsection, we investigate the size of the $GSSKY$. Figure 3 illustrates the $GSSKY$ size on SYN, CA and US datasets where m varies from 2 to 5. For given m, the size difference of the three datasets are not significant. As expected, the number of $GSSKY$ objects increases quickly towards the number of types (m). Particularly, for $m = 2$ the $GSSKY$ size is 13, 15 and 15 for SYN, CA and US respectively. When m goes to 5, it becomes 2, 666, 5, 508 and 5, 911 respectively.

Figure 4 and Figure 5 investigate the impact of the object size (n) and facility size (n_f) respectively. Since locations of the facilities with different types are independent, Theorem 2 can be applied to estimate the $GSSKY$ size , and its accuracy is verified in both Figures. Moreover, the $GSSKY$ size increases slowly with the number of objects and is independent to the number of facilities.

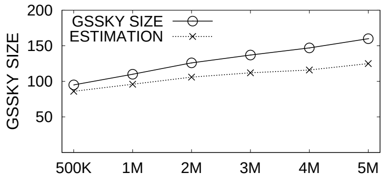

Fig. 4. Diff. n

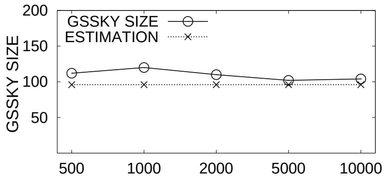

Fig. 5. Diff. n_f

5.2 Efficiency

We first evaluate the efficiency of the algorithms on *SYN*, *CA* and *US* datasets. Figure 6(a) shows that *GSSKY* Algorithm significantly outperforms the *ANN* Algorithm by at least one order of magnitude. The number on each bar records the cost for the skyline test, which shows the dominant cost in two algorithms is the distance computation. We also study the impact of the parameters which may potentially affect the performance of the algorithms. Specifically, Figure 6(b), Figure 6(c) and Figure 6(d) investigate the scalability of the algorithms against the m (#types), n (#objects) and n_f (#facilities each type) respectively. As expected, the performance of the algorithms degrades against the growth of these parameters. Nevertheless, *GSSKY* Algorithm is more scalable than *ANN* Algorithm against n and n_f. As expected, both algorithms are sensitive to m as the *GSSKY* size increases significantly against m.

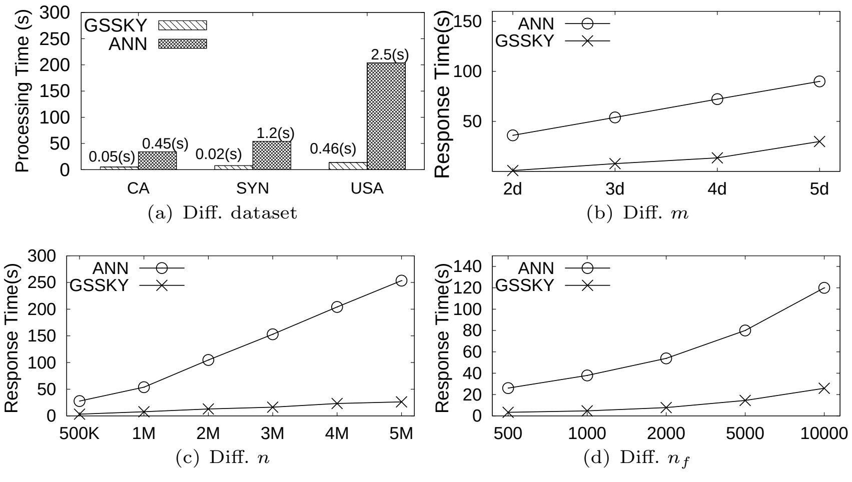

Fig. 6. Time Efficiency Evaluation

In Figure 7, we evaluate the number of R-tree nodes loaded in the main memory on *SYN*,*CA* and *US* datasets. It is shown that *GSSKY* algorithm significantly reduces I/O because many objects are pruned. Figure 8 shows the proportion of the objects involved in distance computation. Clearly, all objects contribute to the distance calculation in *ANN* Algorithm. While a significant number of objects are pruned in *GSSKY* Algorithm. It also demonstrates the accuracy of the Theorem 4, where *EST* represents the estimation of the theorem.

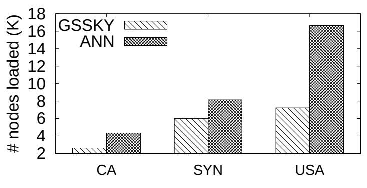

Fig. 7. I/O evaluation

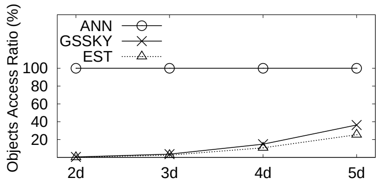

Fig. 8. object accessed vs m

6 Related Work

Studies on skyline computation have a long history. Börzsönyi *et al.* [1] first investigate the skyline computation problem in the context of databases and propose an SQL syntax for the skyline query. They also develop skyline computation techniques based on *block-nested-loop* and *divide-conquer* paradigms, respectively. Chomicki *et al.* [3] propose another block-nested-loop based computation technique, SFS (*sort-filter-skyline*), to take advantages of a pre-sorting. Papadias *et al.* [11] propose a branch and bound search technique (BBS) to progressively output skyline points on dataset indexed by R-tree.

The problem of spatial skyline is first proposed in [14]. Given a set $\mathcal{O}$ of objects and a set $\mathcal{Q}$ of query points, each object has $|\mathcal{Q}|$ derived spatial attributes each of which is the distance of the object to a query point in $\mathcal{Q}$, and hence can be mapped to a point in $|\mathcal{Q}|$-dimensional space where $|\mathcal{Q}|$ is the number of query points in $\mathcal{Q}$. Then the spatial skyline regarding $\mathcal{O}$ and $\mathcal{Q}$ is the traditional skyline on $|\mathcal{Q}|$-dimensional space. Efficient algorithms are developed in [14] to compute spatial skylines by utilizing the R-tree, convex hull, and voronoi diagram techniques. Son *et al.* [15] further improve the spatial skyline computation techniques. Recently, in [16] they investigate the problem based on the manhattan distance. In [4], Ke *et al.* investigate the problem in the road network. Besides the spatial skyline, there are also some related works in which the skyline is computed based on the derived spatial attributes. In [8], Huang *et al.* studies the problem of in-route skyline to find locations which are not dominated by other candidate locations regarding the network distance to a query location q and the corresponding detour distance. In [9, 6] spatial distance regarding a query point q is considered during the skyline computation, in which other dimensions of an objects are non-spatial attributes.

In many applications, the query points may come from the same category (e.g., bus stations, supermarkets). For an object o and a particular category (i.e., facility type like bus station), users are only interested in the distance between o and its closest query point (i.e., facility) in that category. Consequently, the spatial skyline does not make sense in these applications because it considers the distances of o regarding **all** facilities in the same category, and hence cannot provide a minimal set of candidates for the distance based spatial preference queries [10]. Moreover, the techniques in [14, 16] cannot be applied to the *GSSKY* computation because the *spatial skyline* is a special case of the *general spatial skyline* in which there is only one facility for each facility type.

7 Conclusion and Future Work

In this paper, we introduce the *general spatial skyline* which can provide a minimal set of candidates that contain optimal solutions of any monotonic distance based spatial preference query. Efficient algorithm is proposed in the paper and comprehensive experiments are conducted to demonstrate the effectiveness and efficiency of the algorithms. As a possible future work, we will investigate the problem on the road network in which the network distance is considered.

Acknowledgement. The work was supported by ARC grants(DP110104880, DP120104168, DE120102144) and ECR grant(PSE1799)

References

1. S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE*, 2001.
2. Y. Chen and J. M. Patel. Efficient evaluation of all-nearest-neighbor queries. In *ICDE*, 2007.
3. J. Chomicki, P. Godfrey, J. Gryz, and D. Liang. Skyline with presorting. In *ICDE*, 2003.
4. K. Deng, X. Zhou, and H. T. Shen. Multi-source skyline query processing in road networks. In *ICDE*, 2007.
5. P. Godfrey. Skyline cardinality for relational processing. In *FoIKS*, 2004.
6. X. Guo, Y. Ishikawa, and Y. Gao. Direction-based spatial skylines. In *MobiDE*, 2010.
7. G. R. Hjaltason and H. Samet. Distance browsing in spatial databases. *TODS*, 24(2):265–318, 1999.
8. X. Huang and C. S. Jensen. In-route skyline querying for location-based services. In *W2GIS*, 2004.
9. K. Kodama, Y. Iijima, X. Guo, and Y. Ishikawa. Skyline queries based on user locations and preferences for making location-based recommendations. In *GIS-LBSN*, 2009.
10. Q. Lin, C. Xiao, M. A. Cheema, and W. Wang. Finding the sites with best accessibilities to amenities. In *DASFAA*, 2011.
11. D. Papadias, Y. Tao, G. Fu, and B. Seeger. An optimal and progressive algorithm for skyline queries. In *SIGMOD 2003*.
12. J. B. Rocha-Junior, A. Vlachou, C. Doulkeridis, and K. Nørvåg. Efficient processing of top-k spatial preference queries. *PVLDB*, 4(2), 2010.
13. N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *SIGMOD Conference*, 1995.
14. M. Sharifzadeh and C. Shahabi. The spatial skyline queries. In *VLDB*, 2006.
15. W. Son, M.-W. Lee, H.-K. Ahn, and S. won Hwang. Spatial skyline queries: An efficient geometric algorithm. In *SSTD*, 2009.
16. W. Son, S. won Hwang, and H.-K. Ahn. Mssq: Manhattan spatial skyline queries. In *SSTD*, 2011.
17. M. L. Yiu, X. Dai, N. Mamoulis, and M. Vaitis. Top-k spatial preference queries. In *ICDE*, 2007.