# A Skylining Approach to Optimize Influence and Cost in Location Selection

Juwei Shi[1], Hua Lu[2], Jiaheng Lu[1], and Chengxuan Liao[1]

[1] School of Information and DEKE, MOE in Renmin University of China
[2] Department of Computer Science, Aalborg University, Denmark
juwei.shi@gmail.com,luhua@cs.aau.dk,
jiahenglu@ruc.edu.cn,liaochengxuan@gmail.com

**Abstract.** Location-selection problem underlines many spatial decision-making applications. In this paper, we study an interesting location-selection problem which can find many applications such as banking outlet and hotel locations selections. In particular, given a number of spatial objects and a set of location candidates, we select some locations which maximize the influence but minimize the cost. The influence of a location is defined by the number of spatial objects within a given distance; and the cost of a location is indicated by the minimum payment for such location, which is measured by quality vectors. We show that a straightforward extension of a skyline approach is inefficient, as it needs to compute the influence and cost for all the location candidates relying on many expensive range queries. To overcome this weakness, we extend the Branch and Bound Skyline (BBS) method with a novel spatial join algorithm. We derive influence and cost bounds to prune irrelevant R-tree entries and to early confirm part of the final answers. Theoretical analysis and extensive experiments demonstrate the efficiency and scalability of our proposed algorithms.

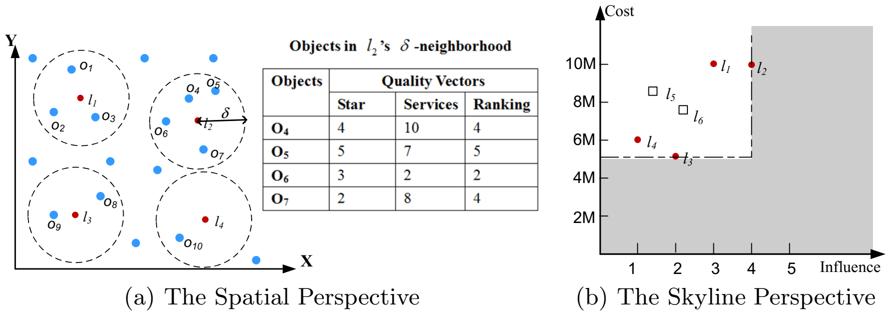
## 1 Introduction

Location selection has been an emerging problem with many commercial applications. For example, telecom service providers store huge volumes of location data to provide data monetization applications to the third party, such as banking outlet and hotel locations selections. In many scenarios, additional attributes besides the location are available in a spatial object. For example, a hotel has a spatial position as well as quality attributes such as star, service, etc. These attributes can improve the price-performance of selected locations. Unfortunately, traditionally location selections only take the spatial distance into account [7, 22, 25], which ignore non-spatial attributes.

In this paper, we select locations in terms of both spatial distances and quality vectors. In particular, we select locations to maximize their *influences* but minimize their *costs*. Given a distance threshold $\delta$, a location $l$'s *influence* is measured by the number of existing spatial objects within the distance $\delta$ from $l$. It indicates how many objects the location can potentially influence. As shown in Figure 1(a), there are four location candidates ($l_1$, $l_2$, $l_3$ and $l_4$) and a bunch of existing spatial objects. Here $l_2$ impacts the most number of objects since its $\delta$-neighborhood contains four existing spatial objects.

Given a distance threshold $\delta$, a location $l$'s *cost* is measured by the payment for obtaining minimal quality vectors [1] to *dominate* all the existing objects within the distance $\delta$ from $l$. The concept of *dominance* [3] is proposed to compare two quality vectors. One quality vector $v_i$ dominates another vector $v_j$ if $v_i$ is no worse than $v_j$ on all attributes and better than $v_j$ on at least one attribute. As shown in Figure 1(a), the minimal quality vector to dominate all the objects in $l_2$'s $\delta$-neighborhood is $\langle 5, 10, 5\rangle$. Furthermore, we assume there is a monotonic function which maps a quality vector to a numerical cost. For instance, the cost of $l_2$ can be defined as $f(l_2) = f(\langle 5, 10, 5\rangle) = \frac{1}{2} \cdot (5 + 10 + 5) = 10$.

In this paper, we adopt the skyline query [3] to define our location selection problem. Given a set of objects, the skyline operator returns a subset of objects such that the object in the subset is not dominated by any other objects. In particular, we select locations whose influence and cost are not dominated by any other locations. The skyline points of locations in Figure 1(a) are shown in Figure 1(b). Suppose that the influence and cost are $\langle 3, 10\rangle$ for $l_1$, $\langle 4, 10\rangle$ for $l_2$, $\langle 2, 5\rangle$ for $l_3$, and $\langle 1, 6\rangle$ for $l_4$. Then $\langle l_2, l_3\rangle$ is the result of the skyline location selection.



Objects in $l_2$'s $\delta$-neighborhood

| Objects | Quality Vectors | | |
|---|---|---|---|
| | Star | Services | Ranking |
| $O_4$ | 4 | 10 | 4 |
| $O_5$ | 5 | 7 | 5 |
| $O_6$ | 3 | 2 | 2 |
| $O_7$ | 2 | 8 | 4 |

(a) The Spatial Perspective (b) The Skyline Perspective

**Fig. 1.** The Example in the Spatial and Skyline Perspective

A straightforward solution to address the skyline location selection problem is that we compute the influence and the cost for each of the location candidates, and then use existing skyline query algorithms to generate the skyline points. For large data sets this is infeasible since it relies on expensive range queries to compute the influence and the cost for all the locations.

We develop an efficient algorithm to answer a skyline location selection query. First, we build two R-trees on the location set $L$ and the object set $O$, denoted

[1] Without loss of generality, we assume that larger values are preferred in the dominance comparison throughout the paper.

as $R_L$ and $R_O$, respectively. At high-level, the algorithm descends the two R-trees in the branch and bound manner, progressively joining $R_L$ entries with $R_O$ entries to compute the two bounds of the influence and the cost for each entry $e_L$ in $R_L$; then based on the generated skyline points, the algorithm decides whether to prune an entry $e_L$, or to access its children until all leaf entries of $R_L$ are accessed. During the two R-trees traversal, we use a min-heap to control the accessing order of $R_L$ entries, which can ensure that an irrelevant R-tree node is not visited before its dominance skyline point is generated.

The contributions of this paper are summarized as follows.

– We define a type of optimal location selection problem that takes into account not only spatial distance but also non-spatial quality vectors associated to locations. Our problem definition returns optimal locations that can potentially influent the largest number of objects in proximity at the lowest costs. Our approach employs the skyline dominance concept that is popular for multi-criteria optimization, and therefore it requires no specific user intervention in selecting optimal locations.
– We propose a novel location selection algorithm. Tight influence and cost bounds are derived to prune irrelevant R-tree entries and to early confirm part of the final answers.
– We provide theoretical analysis on the IO cost of our algorithm based on a spatial join cost model.
– We conduct extensive experiments to evaluate our algorithm under various settings.

The remainder of this paper is organized as follows. Section 2 gives the definitions and the problem statement. Section 3 proposes our algorithm for the problem. Section 4 provides the IO cost analysis of our algorithm. Section 5 evaluates our proposed algorithm experimentally. Section 6 reviews related works. Finally, section 7 concludes the paper and discusses the future work.

## 2 Problem Definition

In this section, we formally define the location selection problem. A location is a point $\lambda = \langle x, y \rangle$ where $x$ and $y$ are coordinate values in a 2-dimensional space. We assume that there are $c$ quality dimensions and $\mathcal{D} = D_1 \times D_2 \times \ldots \times D_c$ is the quality space. A quality vector is a $c$-dimensional point $p = (d_1, d_2, \ldots, d_c)$, where $d_i \in D_i (1 \leq i \leq c)$. Then, a spatial object $o$ is composed of a location $\lambda$ and a quality vector $\psi$ associated with that location, i.e., $o = \langle \lambda, \psi \rangle$. We use $o.loc$ and $o.p$ to denote a location object $o$'s location and quality vector respectively. For a spatial object set $O$, we use $\pi_P(O)$ to denote all the quality vectors associated with spatial objects in $O$, i.e. $\pi_P(O) = \{o.p \mid o \in O\}$. We define the key components of our problem as follows.

**$\delta$-Neighborhood:** Given a location *loc*, a spatial object set $O$, and a distance threshold $\delta$, *loc*'s *$\delta$-Neighborhood*, termed as $N_\delta(loc, O)$, is the subset of objects in $O$ that are within the distance $\delta$ from $loc$. Formally, $N_\delta(loc, O) = \{o \mid o \in O \wedge \|o.loc, o\| \leq \delta\}$. The cardinality of a location's $\delta$-Neighborhood indicates the location's potential influence.

**Quality Dominance:** Given two quality vectors $p$ and $p'$, $p$ dominates $p'$ if $p$ is no worse than $p'$ on all attributes and $p$ is better than $p'$ on at least one attribute. We use $p \prec p'$ to denote that $p$ dominates $p'$. Given a quality vector $p$, there can be multiple quality vectors that dominate $p$. Each of them is called $p$'s quality dominator. Given a quality vector set $P$, we use $p \prec P$ to indicate that $p$ is $P$'s dominator.

**Cost Functions:** Given a quality vector $q \in \mathcal{D}$, the cost function $f^g_{cost}(q)$ returns a cost value of real type, i.e. $f^g_{cost} : \mathcal{D} \rightarrow \mathcal{R}$. For example, a cost function can be defined as the weighted sum of all quality attribute values, i.e., $f^c_{cost}(q) = \sum_{i=1}^{c} w_i \cdot q.d_i$. Here, $w_i$ is zero if quality dimension $D_i$ has no impact on the dominance cost; $w_i > 0$ if the dominance cost is proportional to the values on quality dimension $D_i$; otherwise $w_i < 0$. Note that it is natural to define cost functions that are monotonic with respect to dominance. We say a cost function $f_{cost}$ is *monotonic* if and only if $f_{cost}(q) \geq f_{cost}(q')$ for any two quality vectors that satisfy $q \prec q'$. This is consistent with the intuition that better quality is achieved at a higher cost.

**Minimum Quality Dominance Cost:** Among all the quality dominators, we are interested in the one with the minimum cost. Given a quality space $\mathcal{D}$, a set of quality vectors $P \subseteq \mathcal{D}$, and a cost function $f_{cost}$, we use $\mathbb{D}(P)$ to denote $P$'s minimum cost quality dominator such that

1. $\mathbb{D}(P) \prec P$;
2. $\forall p' \in \mathcal{D}$ and $p' \prec P$, $f_{cost}(p') \geq f_{cost}(\mathbb{D}(P))$.

Then, we define $\mathbb{C}(P) = f_{cost}(\mathbb{D}(P))$ to denote $P$'s *Minimum Quality Dominance Cost.*

Note that $\mathbb{D}(P)$ is a quality vector and $\mathbb{C}(P)$ is a scalar value. As we assume large values are preferred in the dominance comparison, $\mathbb{D}(P).d_i = \min\{(p.d_i) \mid p \prec P\}$.

**Location Dominance:** Given a spatial object set $O$, a distance threshold $\delta$, and a cost function $f_{cost}$, a location $loc_1$ *location dominates* another location $loc_2$, termed as $loc_1 \prec loc_2$, if and only if

1. $|N_\delta(loc_1, O)| \geq |N_\delta(loc_2, O)|$;
2. $\mathbb{C}(\pi_P(N_\delta(loc_1, O))) \leq \mathbb{C}(\pi_P(N_\delta(loc_2, O)))$;
3. $|N_\delta(loc_1, O)| > |N_\delta(loc_2, O)|$ or $\mathbb{C}(\pi_P(N_\delta(loc_1, O))) < \mathbb{C}(\pi_P(N_\delta(loc_2, O)))$.

With all definitions formulated above, we give the problem statement of optimal location selection as follows.

*Problem 1.* Given a location set $L$, a spatial object set $O$, a distance threshold $\delta$, and a cost function $f_{cost}$, an optimal location selection returns from $L$ a subset of locations that are not location dominated by any others. Formally, the Optimal Location Selection (OLS) problem is defined as:

$$OLS(L, O, \delta) = \{l \mid l \in L, \nexists l' \in L \wedge l' \prec l\} \tag{1}$$

## 3 Algorithms for Optimal Location Selection

In this section, we present algorithms for defined optimal location selection. We start from the naive loop algorithm, and then describe the spatial join based algorithm.

**Algorithm 1. Loop**(Spatial object set $O$'s R-tree $R_O$, location set $L$, distance threshold $\delta$)

```
1:  $S \leftarrow \emptyset$
2:  for each location $l \in L$ do
3:      $N_\delta(l, O) \leftarrow$ Range_Query$(l, \delta, R_O)$
4:      $p \leftarrow (0, \dots 0)$                                  $\triangleright$ $c$-dimensional point
5:      for each object $o \in N_\delta(l, O)$ do
6:          $p[i] \leftarrow \max(p[i], o.p[i])$
7:      $cost \leftarrow f_{cost}(p)$                                $\triangleright$ $\mathbb{C}(\pi_P(N_\delta(l, O)))$
8:      $influence \leftarrow |N_\delta(l, O)|$
9:      $candidate \leftarrow (l, influence, cost)$
10:     dominanceCheck$(S, candidate)$
11: return $S$
```

**Algorithm 2. dominanceCheck**(Current skyline $S$, a candidate *candidate*)

```
1: $flag \leftarrow$ false
2: for each tuple $tp \in S$ do
3:     if $tp \prec candidate$ then
4:         $flag \leftarrow$ true;  break
5:     else if $candidate \prec tp$ then
6:         remove $tp$ from $S$
7: if $flag =$ false then
8:     add $candidate$ to $S$
```

### 3.1 The Loop Algorithm

We develop a loop algorithm shown in Algorithm 1 as the baseline algorithm. The idea is that: for each location $l \in L$, we issue a range query on the object set $O$ to get $l$'s $\delta$-neighborhood $N_\delta(l, O)$. All objects in the $\delta$-neighborhood are checked to obtain the minimum cost quality dominator. Then, the influence and cost of a location candidate are computed. Finally, the candidate is checked against all generated optimal locations in terms of location dominance. This dominance check is shown in Algorithm 2.

## 3.2 The Join Algorithm

To reduce the computation overhead of influence and cost, we propose a spatial join based algorithm. The basic idea is that we make use of the R-tree [9] based spatial join [4] to find the $\delta$-neighborhood for location candidates.

Suppose that spatial attributes of the location set $L$ and the object set $O$ are indexed by the R-trees $R_L$ and $R_O$ respectively. The locations in $L$ are joined with the objects in $O$ based on the two R-trees: an entry $e_L$ from $R_L$ are joined with a set of overlapped entries from $R_O$. These relevant entries are defined as $e_L$'s *join list*. We use $e_L.JL$ to denote $e_L$'s join list. Intuitively, we avoid to find the joint list from the whole object data set, and make use of the spatial join to obtain the joint list only from relevant object R-tree entries. Thus the IO cost of operations on the object set $O$ is significantly reduced.

To further reduce the overhead of influence and cost computations for irrelevant location candidates, we derive the influence bound and the cost bound for all locations in a given location R-tree entry $e_L$. The two bounds are used in the join algorithm to prune irrelevant $R_L$ and $R_O$ nodes.

**Influence Bound.** We introduce the $\delta$-Minkow-ski region [2] to derive the influence bound. Given a distance threshold $\delta$, a location entry $e_L$'s $\delta$-Minkow-ski region, denoted by $\Xi(e_L, \delta)$, is the set of all locations whose minimum distance from $e_L$ is within threshold $\delta$. Formally, we define

$$\Xi(e_L, \delta) = \{t \in \mathbb{R}^2 \mid dist_{min}(t, e_L) \leq \delta\} \tag{2}$$

We are interested in those objects from $O$ that fall into Region $\Xi(e_L, \delta)$. Accordingly, we define $e_L$'s $\delta$-Minkowski region with respect to $O$ as follows.

$$\Xi_O(e_L, \delta) = \{o \in O \mid o.loc \in \Xi(e_L, \delta)\} \tag{3}$$

Given a spatial object set $O$ and a distance threshold $\delta$, we define the *influence bound* of $e_L$, termed as $BI_{O,\delta}(e_L)$, to be the number of objects in $\Xi_O(e_L, \delta)$.

$$BI_{O,\delta}(e_L) = |\Xi_O(e_L, \delta)| \tag{4}$$

If $e'_L$ is a descent entry of $e_L$, we have $\Xi_O(e'_L, \delta) \subseteq \Xi_O(e_L, \delta)$. Therefore, we have the following lemma that guarantees the correctness of the influence bound.

**Lemma 1.** *Given a spatial object set $O$, a distance threshold $\delta$, and a location entry $e_L$, any location $l$'s influence cannot be larger than $e_L$'s influence bound, i.e. $|N_\delta(l, O)| \leq BI_{O,\delta}(e_L)$.*

*Proof.* The lemma is proved by the fact that $\forall l \in e_L$, $N_\delta(l, O) \subseteq \Xi_O(e_L, \delta)$.

**Cost Bound.** Given a spatial object set $O$ and a distance threshold $\delta$, we term the *cost bound* of $e_L$ as $BC_{O,\delta}(e_L)$. Intuitively, $BC_{O,\delta}(e_L)$ can be the cost to dominate the most disadvantaged quality vector among all that are associated to locations in $e_L$. Since we assume large values are preferred in the dominance comparison, the most disadvantaged quality vector is the one that has the minimum value on all attributes. Then, that (virtual) quality vector is defined as $mdq(\Xi_O(e_L, \delta))$ where $mdq.d_i = \min\{p.d_i \mid p \in \pi_P(\Xi_O(e_L, \delta))\}$.

Given a quality cost function $f_{cost}$, we define the cost bound $BC_{O,\delta}(e_L)$ as the cost to dominate this most disadvantaged quality vector.

$$BC_{O,\delta}(e_L) = f_{cost}(mdq(\Xi_O(e_L, \delta))) \tag{5}$$

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

**Lemma 2.** *Given a spatial object set $O$, a distance threshold $\delta$, and an location entry $e_L$, $e_L$'s cost bound is larger than or equal to that of its any descent entry $e'_L$, i.e. $BC_{O,\delta}(e'_L) \geq BC_{O,\delta}(e_L)$.*

*Proof.* Suppose the most disadvantaged quality vectors in $\Xi_O(e_L, \delta)$ and $\Xi_O(e'_L, \delta)$ are $mdq$ and $mdq'$ respectively. We have $mdq.d_i = \min\{p.d_i \mid p \in \pi_P(\Xi_O(e_L, \delta))\}$ and $mdq'.d_i = \min\{p.d_i \mid p \in \pi_P(\Xi_O(e'_L, \delta))\}$. Since $e'_L \subseteq e_L$, we have $\Xi_O(e'_L, \delta) \subseteq \Xi_O(e_L, \delta)$. Therefore, we have $mdq'.d_i \geq mdq.d_i$. Due to the monotonicity of the quality cost function $f_{cost}$, we have $f_{cost}(mdq') \geq f_{cost}(mdq)$, i.e., $BC_{O,\delta}(e'_L) \geq BC_{O,\delta}(e_L)$.

As a remark, the cost of a location in the entry $e_L$ satisfies $\mathbb{C}(\pi_P(N_\delta(l, O))) = f_{cost}(\mathbb{D}(\pi_P(N_\delta(l, O)))) \geq BC_{O,\delta}(e_L)$.

**The Join Algorithm.** We propose the join algorithm in Algorithm 3 and Algorithm 4. To make use of the influence bound, we index the object set $O$ with an aggregate R-tree $R_O$ in which each non-leaf node entry $e$ has an additional filed *e.count*. Here *e.count* is the total number of all spatial objects in $e$. Similarly, to make use of the cost bound, each non-leaf node entry $e$ in $R_O$ has another additional filed $e.\psi$, which is a quality vector defined as follows.

$$e.\psi.d_i = \min\{o.p.d_i \mid o \in e\} \tag{6}$$

Thus, we extend each non-leaf node entry $e$ in object R-tree $R_O$ with two extra fields *e.count* and $e.\psi$. Since the calculation of the two bounds is in the course of the spatial join, no additional IO costs on $R_O$ and $R_L$ are introduced. Accessing all location entries in $R_L$ is prioritized by a min-heap. A location entry $e_L$ is pushed to the heap with a key which equals to $BC_{O,\delta}(e_L) - \sum_{e \in e_L.JL} e.count$, i.e. the difference between the influence and cost bound. When the value of two keys are the same, we randomly select one entry as the lower value key. The min-heap ensures that irrelevant R-tree nodes will not be visited before its dominance skyline point is generated.

As a remark, our algorithm follows the spirit of the well-established Branch-and-Bound Skyline (BBS) algorithm [15] that prioritizes R-tree node access to ensure that skyline points are always generated before their dominating R-tree nodes are visited. The difference is that we integrate the branch-and-bound to the spatial join algorithm such that the $\theta$-neighborhood of a location can be efficiently found.

**Algorithm 3. Join_Together**(Spatial object set $O$'s combined R-tree $R_O$, location set $L$'s R-tree $R_L$, distance threshold $\delta$)

```
1: $S \leftarrow \emptyset$
2: initialize a min-heap $H$
3: $e_{root} \leftarrow R_L.root$; $e_{root}.JL \leftarrow \{R_O.root\}$
4: enheap($H, \langle e_{root}, e_{root}.JL, 0, 0, 0\rangle$)
5: while $H$ is not empty do
6:     $\langle e_L, e_L.JL, count, cost, v\rangle \leftarrow$ deheap($H$)
7:     if $\exists tp \in S$ s.t. $tp \prec (*, count, cost)$ then
8:         continue
9:     if $e_L$ is a leaf entry then
10:        $l \leftarrow$ the location pointed by $e_L$
11:        $influence \leftarrow count$
12:        $candidate \leftarrow (l, influence, cost)$
13:        add $candidate$ to $S$
14:    else
15:        read the child node $CN_L$ pointed to by $e_L$
16:        for each entry $e_i$ in $CN_L$ do
17:            $count \leftarrow 0$; $e_i.JL \leftarrow \emptyset$
18:            for each $e_j$ in $e_L.JL$ do
19:                if $\Xi(e_i, \delta)$ contains $e_j$ then
20:                    add $e_j$ to $e_i.JL$;  $count \leftarrow count + e_j.count$
21:                else
22:                    read the child node $CN_O$ pointed to by $e_j$
23:                    Minkowski($e_i, \delta, CN_O, e_i.JL, count$)
24:            for each entry $e \in e_i.JL$ do
25:                $p[i] \leftarrow \min(p[i], e.\psi[i])$
26:            $cost \leftarrow f_{cost}(p)$
27:            if $\exists tp \in S$ s.t. $tp \prec (*, count, cost)$ then
28:                continue
29:            else
30:                enheap($H, \langle e_i, e_i.JL, count, cost, cost - count\rangle$)
31: return $S$
```

**Algorithm 4. Minkowski**(R-tree $R_L$'s entry $e_L$, distance threshold $\delta$, aggregate R-tree $R_O$'s node $CN_O$, aggregate R-tree $R_O$'s entry list $JL$, count $v$)

```
1: for each child entry $e$ in $CN_O$ do
2:     if $\Xi(e_L, \delta)$ contains $e$ then
3:         add $e$ to $JL$;  $v \leftarrow v + e.count$
4:     else
5:         read the child node $CN_P$ pointed to by $e$
6:         Minkowski($e_L, \delta, CN_P, JL, v$)
```

## 4 Analysis

In this section, we first prove the correctness of the proposed algorithm. Then, we provide IO cost analysis for our algorithm.

### 4.1 Correctness of the Algorithm

The proof of the correctness is similar to that proposed in [15]. We use $B_i$ and $B_c$ to denote the influence and cost bound respectively. The difference is that our algorithm visits entries of the location R-tree $R_L$ in ascending order based on the distance between $\langle B_i, B_c \rangle$ and $\langle \infty, 0 \rangle$ on the influence-cost formed coordinate plane. It is straightforward to prove that our algorithm never prunes a location entry of $R_L$ which contains skyline points.

### 4.2 IO Cost Analysis

To quantify the IO cost of the proposed algorithm, we extend the concept of Skyline Search Region (SSR) proposed by [15]. In this paper, the SSR is the area defined by the skyline points and the two axes of influence and cost. For example, the SSR area is shaded in Figure 1(b). Our algorithm must access all the nodes whose $\langle B_i, B_c \rangle$ falls into the SSR. In other word, if a node does not contain any skyline points but its $\langle B_i, B_c \rangle$ falls into SSR, it will also be visited if it has not been pruned.

**Lemma 3.** *If the influence and cost bound of an object entry $e$ does not intersect the SSR, then there is a skyline point $p$ whose distance to $\langle \infty, 0 \rangle$ is smaller than the distance between $e$ and $\langle \infty, 0 \rangle$.*

*Proof.* Since the influence and cost bounds of the object R-tree entry dominate that of all its child node, $p$ dominates all the leaf nodes covered by $e$.

**Theorem 1.** *An entry of the location R-tree will be pruned if its influence and cost bounds $\langle B_i, B_c \rangle$ fall into the SSR.*

*Proof.* We prove it based on Lemma 3 and the fact that we visit $R_L$ in the order of $B_i - B_c$. Based on the min-heap structure, if there is a skyline point that dominates the entry bounded by $\langle B_i, B_c \rangle$, the skyline point will be visited earlier than that entry. Thus the entry will be pruned when it is popped up from the heap.

Next, we derive IO cost of the Join_Together algorithm based on the cost model proposed in [29]. Let $P_L(i)$ be the probability that a level $i$ node's $\langle B_i, B_c \rangle$ is contained by the SSR. The number of node accesses at the $i$th level of the location R-tree $R_L$ equals:

$$NA_L(i) = \frac{N_L}{f_L^{i+1}} \cdot P_L(i) \tag{7}$$

where $N_L$ is the cardinality of the location candidate data set $L$ and $f_L$ is the node fan-out of $R_L$. Let $P_L(\alpha, \beta, i)$ be the probability that $\langle B_i, B_c \rangle$ of a level $i$ node of $R_L$ is contained by the rectangle with the corner points $\langle \infty, 0 \rangle$ and $\langle \alpha, \beta \rangle$. The density of the influence and cost equals:

$$D_L(\alpha, \beta, i) = \frac{\partial^2 P(\alpha, \beta, i)}{\partial \alpha \partial \beta} \tag{8}$$

Then we have

$$P_L(i) = \int\int_{\langle x,y\rangle \in SSR} D_L(x, y, i)dxdy \tag{9}$$

where $x$ and $y$ is the influence and cost bounds of $R_L$ entries respectively. Thus we obtain the IO cost of location R-tree:

$$NA_L = \sum_{i=1}^{h_L-1} NA_L(i) \tag{10}$$

where $h_L$ is the height of the location R-tree $R_L$.

**Lemma 4.** *To get the join list of an entry in $R_L$, we only expand the join list of its parent.*

*Proof.* The lemma is proved by the fact that the join list of a entry is computed based on its parent's join list from the heap.

Let $P_O(j)$ be the probability that a level $j$ node from $R_O$ intersects with unpruned entries from $R_L$, we have

$$NA_O(j) = \frac{N_O}{f_O^{j+1}} \cdot P_O(j) \cdot f_L \tag{11}$$

where $N_O$ is the cardinality of the object data set $O$ and $f_O$ is the node fan-out of $R_O$, and we have

$$P_O(i) = \int\int_{\langle x,y\rangle \in L_{unpruned}} D_L(x, y, i)dxdy \tag{12}$$

where $\langle x, y\rangle \in L_{unpruned}$ denotes the location of a level $j$ node intersects with an unpruned $R_L$ node. Thus we obtain the IO cost of the object R-tree $R_O$:

$$NA_O = \sum_{j=1}^{h_O-1} NA_O(j) \tag{13}$$

where $h_O$ is the height of the location R-tree $R_O$.

Finally, the number of node accesses of both $R_L$ and $R_O$ equals

$$NA = NA_L + NA_O \tag{14}$$

## 5 Experimental Studies

### 5.1 Settings

We use both real and synthetic data sets in our experiments. The real world US hotel (USH) data set consists of 30,918 hotel records with the schema `(longitude, latitude, review, stars, price)`. For all hotel records, their locations (longitude and latitude) are normalized to the domain $[0, 10000] \times [0, 10000]$, and their quality attributes are normalized to the domain $[0, 1]^3$. We perform value conversions on quality attributes to make smaller values preferable. We randomly extract 918 hotels from USH and use their locations as the location set $L$. The remaining 3000 hotels form the object set $O$.

We also generate an object set with three independent quality attributes, and another object set with three anti-correlated quality attributes. All quality attribute values are normalized to the range [0, 1]. Both object sets contain 100,000 objects whose locations are randomly assigned within the space $[0, 10000] \times [0, 10000]$. As larger quality attribute values are preferred in our setting, we employ a cost function $f^c_{cost}(q) = \sum_{i=1}^{c} q.d_i$ in all experiments.

We set the page size to 4 KB when building the R-trees. All trees have node capacities between 83 and 169. All algorithms are implemented in Java and run on a Windows platform with Intel Core 2 CPU (2.54GHz) and 2.0 GB memory.

## 5.2 Performance of Location Selection Algorithms

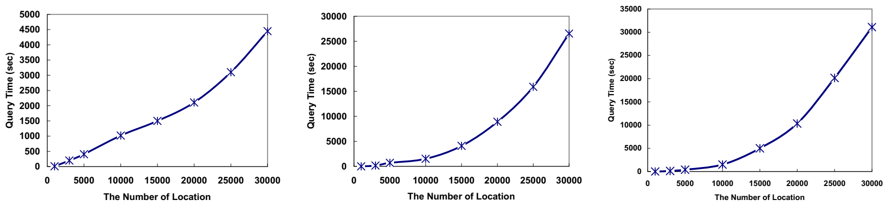
We report an experimental evaluation of skyline location selection algorithms, namely Loop (Algorithm 1) and Join_Together (Algorithm 3). To study the effect of each bound separately, we add Join_Influence and Join_Cost which use either the influence or the cost bound only.

In the Join_Influence algorithm, each non-leaf Object R-tree entry $e$ has an extra filed $e.count$ that is the total number of all spatial objects in $e$. Accessing all location entries in $R_L$ is prioritized by a max-heap with a key which equals to the influence bound $\sum_{e \in e_L.JL} e.count$. Similarly, in the Join_Cost algorithm, an extra filed $e.\psi$ is added to the object R-tree entry $e$. Here $e.\psi$ is defined as $e.\psi.d_i = \min\{o.p.d_i \mid o \in e\}$. $e_L$ is pushed to a min-heap with a key which equals to the cost bound $BC_{O,\delta}(e_L)$.

**The Impact of the Number of Query Locations:** In order to study the impact of the number of query locations, we vary the number of query locations in $L$. All locations in each $L$ are generated at random with the spatial domain $[0, 10000] \times [0, 10000]$. We set the distance threshold $\delta$ to 800.

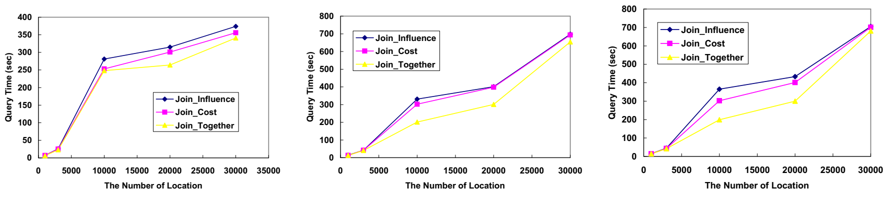
Figure 2 and Figure 3 show the results of the loop and the join algorithms, respectively. The join algorithms are significantly more efficient than the loop algorithm under each setting in the experiments. Because the join algorithms leverage the R-tree based spatial join to prune many irrelevant nodes. Figure 2 and Figure 3 indicate that the response time of the skyline location selection increases as the number of locations increases for all the algorithms. Figure 3 indicates that the Join_Together performs better than Join_Influence and Join_Cost. This is because that Join_Together makes use of both bounds to prune more irrelevant nodes.

**The Impact of the Distance Threshold $\delta$:** Next, we evaluate the impact of distance threshold $\delta$ for all the join algorithms. We use 3000 locations on both real and synthetic data sets. We vary $\delta$ using 80, 400, 800, 2000 and 5000.



(a) 30k real USH (b) 100k independent (c) 100k anti-correlated

**Fig. 2.** The Loop Algorithm Performance ($\delta$= 800)



(a) 30k real USH (b) 100k independent (c) 100k anti-correlated

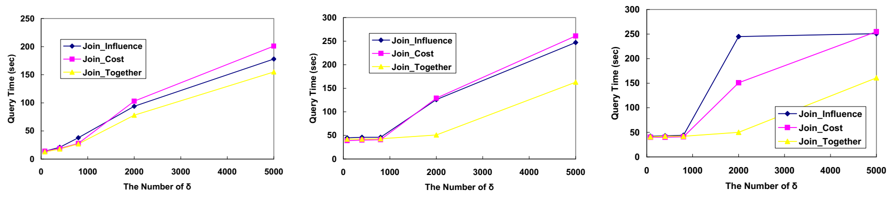
**Fig. 3.** The Join Algorithm Performance ($\delta$= 800)

The results shown in Figure 4 indicate that the response time of skyline location selection increases when the distance threshold $\delta$ is increased. Because a larger distance indicates that more spatial objects will be involved in the spatial join and subsequent checks.

**The Impact of the Query Location Coverage Area:** Finally, we evaluate the impact of the query location coverage area, i.e., the region of all query locations in $L$. We set the number of query locations to 3000, and the distance threshold $\delta$ to 800. We first use a set of small query location coverage areas that varies from 0.4% to 8.0% of the entire space of interest. The result is shown in Figure 5(a). The Join_Cost outperforms the other two algorithms when all query locations are distributed in a very small part of the entire space. It indicates that the cost bound is more effective than the influence bound when all query locations are very close. When locations are close, their Minkowski regions tend to overlap intensively, which weakens the influence bound based pruning that counts on the number of objects in Minkowski regions.

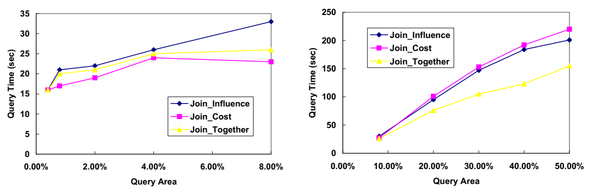
We also use a set of large query location coverage areas that vary from 8% to 50% of the entire space. The result is shown in Figure 5(b). We see that the Join_Together outperforms the other two algorithms. When the query locations cover a larger area, there is less overlap among their Minkowski regions. Then, the influence bound become more effective. Therefore, the combination of both bounds performs the best among all algorithms.

**Summary:** The experimental results show that our proposed spatial join algorithms outperforms the baseline method in the skyline location selection. The



(a) 30k real USH (b) 100k independent (c) 100k anti-correlated

**Fig. 4.** The Effect of $\delta$ (3000 query locations)



(a) Query with small area (b) Query with large area

**Fig. 5.** The Query Area, 3000 locations, $\delta$= 800

combined optimization with the influence and the cost bounds achieves the best performance in most cases in our experiments.

## 6 Related Work

**Spatial Location Selection.** A nearest neighbor (NN) query returns the locations that are closest to a given location. A NN query can be efficiently processed via an R-tree on the location data set, in either a depth-first search [17] or a best-first search [10]. In contrast, the optimal location selection query in this paper considers not only the spatial distances but also quality attributes.

So far in the literature, various constraints have been proposed to extend the nearest neighbor concept to select semantically optimal locations or objects. Du et al. [7] proposed the optimal-location query which returns a location with maximum influence. Xia et al. [22] defined a different top-$t$ most influential spatial sites query, which returns $t$ sites with the largest influences. Within the same context, Zhang et al. [25] proposed the min-dist optimal-location query. However, these proposals do not consider quality attributes of spatial objects.

**Skyline Queries.** Borzonyi et al. [3] defined the skyline query as a database operator, and gave two skyline algorithms: *Block Nested Loop* (BNL) and *Divide-and-Conquer* (D&C). Chomicki et al. [5] proposed a variant of BNL called the *Sort-Filter-Skyline* (SFS) algorithm. Godfrey et al. [8] provided a comprehensive analysis of these non-index-based algorithms and propose a hybrid method with improvements. Bartolini et al. [1] proposed a presorting based algorithm that is able to stop dominance tests early. Zhang et al. [26] proposed a dynamic indexing tree for skyline points (not for the data set), which helps reduce CPU costs in sort-based algorithms [1, 5, 8]. None of the above skyline algorithms require any indexing of the data set.

Alternative skyline algorithms require specific indexes. Tan et al. [20] proposed two progressive algorithms: *Bitmap* and *Index*. The former represents points by means of bit vectors, while the latter utilizes data transformation and $B^+$-tree indexing. Kossmann et al. [6] proposed a *Nearest Neighbor* (NN) method that identifies skyline points by recursively invoking $R^*$-tree based depth-first NN search over different data portions. Papadias et al. [15] proposed a *Branch-and-Bound Skyline* (BBS) method that employs an R-tree on the data set. Lee et al. [12] proposed ZB-tree to access data points in Z-order in order to compute/update skylines more efficiently. Recently, Liu and Chan [14] improved the ZB-tree with a nested encoding to further speed up skyline computation. However, these skyline query algorithms do not address the computation overhead of influence and cost of location candidates.

In [27], the authors proposed several efficient algorithms to process skyline view queries in batch to address the recommendation problem. Hu et al. [28] proposed a deterministic algorithm to address the I/O issue of skyline query. However, none of these works can be directly applied to solve the optimal location selection problem proposed in this paper. The Cost Bound has the same principle as the pseudo documents in IR-tree [16], but we use a spatial join to answer a skyline location selection query.

## 7 Conclusion and Future Work

In this paper, we defined a skyline location selection problem to maximize the influence and minimize the cost. We proposed a spatial join algorithm that can prune irrelevant R-tree nodes. We also conducted theoretical analysis about the IO cost of the algorithm. The extensive experiments demonstrated the efficiency and scalability of the proposed algorithm. As for the future works, we are extending the skyline location selection algorithm to achieve the low-cost scalability in a distributed data management framework.

**Acknowledgment.** This paper is partially supported by 863 National High-tech Research Plan of China (No. 2012AA011001), NSF China (No: 61170011), NSSF China (No: 12&ZD220) and Research Funds of Renmin University of China (No.11XNJ003).

## References

1. Bartolini, I., Ciaccia, P., Patella, M.: Efficient sort-based skyline evaluation. ACM Trans. Database Syst. (TODS) 33(4) (2008)
2. Böhm, C.: A cost model for query processing in high dimensional data spaces. ACM Trans. Database Syst. (TODS) 25(2), 129–178 (2000)
3. Börzsönyi, S., Kossmann, D., Stocker, K.: The skyline operator. In: Proc. ICDE, pp. 421–430 (2001)

4. Brinkhoff, T., Kriegel, H.-P., Seeger, B.: Efficient processing of spatial joins using r-trees. In: Proc. SIGMOD, pp. 237–246 (1993)
5. Chomicki, J., Godfrey, P., Gryz, J., Liang, D.: Skyline with presorting. In: Proc. ICDE, pp. 717–719 (2003)
6. Kossmann, D., Ramsak, F., Rost, S.: Shooting stars in the sky: An online algorithm for skyline queries. In: Proc. VLDB, pp. 275–286 (2002)
7. Du, Y., Zhang, D., Xia, T.: The optimal-location query. In: Medeiros, C.B., Egenhofer, M., Bertino, E. (eds.) SSTD 2005. LNCS, vol. 3633, pp. 163–180. Springer, Heidelberg (2005)
8. Godfrey, P., Shipley, R., Gryz, J.: Maximal vector computation in large data sets. In: Proc. VLDB, pp. 229–240 (2005)
9. Guttman, A.: R-trees: A dynamic index structure for spatial searching. In: Proc. SIGMOD, pp. 47–57 (1984)
10. Hjaltason, G., Samet, H.: Distance browsing in spatial databases. ACM Trans. on Database Syst. (TODS) 24(2), 265–318 (1999)
11. Korn, F., Muthukrishnan, S.: Influence sets based on reverse nearest neighbor queries. In: ACM SIGMOD Record, vol. 29, pp. 201–212 (2000)
12. Lee, K.C.K., Zheng, B., Li, H., Lee, W.-C.: Approaching the skyline in z order. In: Proc. VLDB, pp. 279–290 (2007)
13. Li, C., Tung, A.K.H., Jin, W., Ester, M.: On dominating your neighborhood profitably. In: Proc. VLDB, pp. 818–829 (2007)
14. Liu, B., Chan, C.-Y.: Zinc: Efficient indexing for skyline computation. PVLDB 4(3), 197–207 (2010)
15. Papadias, D., Tao, Y., Fu, G., Seeger, B.: An optimal and progressive algorithm for skyline queries. In: Proc. SIGMOD, pp. 467–478 (2003)
16. Cong, G., Jensen, C.S., Wu, D.: Efficient retrieval of the top-k most relevant spatial web objects. PVLDB 2(1), 337–348 (2009)
17. Roussopoulos, N., Kelley, S., Vincent, F.: Nearest neighbor queries. In: ACM SIGMOD Record, vol. 24, pp. 71–79 (1995)
18. Stanoi, I., Agrawal, D., Abbadi, A.: Reverse nearest neighbor queries for dynamic databases. In: ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, pp. 44–53 (2000)
19. Stanoi, I., Riedewald, M., Agrawal, D., El Abbadi, A.: Discovery of influence sets in frequently updated databases. In: Proc. VLDB, pp. 99–108 (2001)
20. Tan, K.L., Eng, P.K., Ooi, B.C.: Efficient progressive skyline computation. In: Proc. VLDB, pp. 301–310 (2001)
21. Tao, Y., Papadias, D., Lian, X.: Reverse knn search in arbitrary dimensionality. In: Proc. VLDB, pp. 744–755 (2004)
22. Xia, T., Zhang, D., Kanoulas, E., Du, Y.: On computing top-t most influential spatial sites. In: Proc. VLDB, pp. 946–957 (2005)
23. Xiao, X., Yao, B., Li, F.: Optimal location queries in road network databases. In: Proc. ICDE, pp. 804–815 (2011)
24. Yang, C., Lin, K.: An index structure for efficient reverse nearest neighbor queries. In: Proc. ICDE, pp. 485–492 (2001)
25. Zhang, D., Du, Y., Xia, T., Tao, Y.: Progressive computation of the min-dist optimal-location query. In: Proc. VLDB, pp. 643–654 (2006)
26. Zhang, S., Mamoulis, N., Cheung, D.W.: Scalable skyline computation using object-based space partitioning. In: Proc. SIGMOD, pp. 483–494 (2009)

27. Chen, J., Huang, J., Jiang, B., Pei, J., Yin, J.: Recommendations for two-way selections using skyline view queries. In: Knowledge and Information Systems, pp. 397–424 (2013)
28. Hu, X., Sheng, C., Tao, Y., Yang, Y., Zhou, S.: Output-sensitive Skyline Algorithms in External Memory. In: Proc. SODA, pp. 887–900 (2013)
29. Theodoridis, Y., Stefanakis, E.: T.K. Sellis Efficient Cost Models for Spatial Queries Using R-Trees. IEEE Trans. Knowl. Data Eng (TKDE) 12(1), 19–32 (2000)