SKY R-tree: An Index Structure for Distance-Based Top-k Query

Yuya Sasaki[1], Wang-Chien Lee[2], Takahiro Hara[1], and Shojiro Nishio[1]

[1] Osaka University, Osaka, Japan
{sasaki.yuya,hara,nishio}@ist.osaka-u.ac.jp
[2] The Pennsylvania State University, PA, USA
wlee@cse.psu.edu

Abstract. Searches for objects associated with location information and non-spatial attributes have increased significantly over the years. To address this need, a top-k query may be issued by taking into account both the location information and non-spatial attributes. This paper focuses on a distance-based top-k query which retrieves the best objects based on distance from candidate objects to a query point as well as other non-spatial attributes. In this paper, we propose a new index structure and query processing algorithms for distance-based top-k queries. This new index, called *SKY R-tree*, drives on the strengths of R-tree and Skyline algorithm to efficiently prune the search space by exploring both the spatial proximity and non-spatial attributes. Moreover, we propose a variant of SKY R-tree, called *S2KY R-tree* which incorporates a similarity measure of non-spatial attributes. We demonstrate, through extensive experimentation, that our proposals perform very well in terms of I/O costs and CPU time.

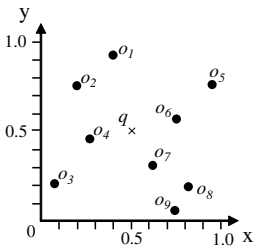
Keywords: Top-k query, Spatial database, R-tree, Skyline, Location-based service.

1 Introduction

With the development of positioning technology and wide availability of mobile devices, efficient location-based search becomes essential for many applications, where both the location information and non-spatial attributes are considered. For example, Yelp and Foursquare[1], two well known location-based social networking services (LBSNs), may employ location-based search to assist their users to find restaurants based on specified location and user preference. Consider Figure 1, where 9 objects $o_1, o_2, \cdots, o_9$ are located in spatial area (illustrated on the left of the figure), and each of them is associated with 2 non-spatial attributes, i.e., price and rating (as illustrated by the table on the right). Given a query point q, a user wants to find the best restaurant o based on both the distance between q and o as well as the user's preference in terms of price and rating of

[1] http://www.yelp.com/,https://foursquare.com/

S.S. Bhowmick et al. (Eds.): DASFAA 2014, Part I, LNCS 8421, pp. 220–235, 2014.



	price	rating
o_1	0.1	1.0
o_2	0.8	0.1
o_3	0.3	0.8
o_4	0.3	0.4
o_5	0.9	0.3
o_6	0.2	0.7
o_7	0.3	0.3
o_8	0.6	0.2
o_9	0.6	0.6

Fig. 1. A spatial area containing objects and a query point

the restaurants.[2] The best object is determined based on the user's preference and the query point.

Several top-k queries for spatial database have been proposed recently [3, 10]. In [10], a *distance-based top-k query* which takes into account both the location and other non-spatial attributes has been proposed. In this query, a user designates the number of requested objects k, a query point q, and a weight α that controls the importance between the roles of distance and other non-spatial attributes in the search. In other words, the distance-based top-k query returns k objects ranked based on "the distance between the objects and q" and "the user preference over the non-spatial attributes". Distance-based top-k queries in [10] retrieve the best k objects based on the branch-and-bound algorithm [9]. However, this branch-and-bound algorithm must construct the tree structure for each distance-based top-k query in the system. Since the paper [10] focuses on a continuous query, it does not care about the cost that constructs the tree structures, but it is absolutely inefficient for a snapshot query. To the best knowledge of the authors, efficient indexing techniques for distance-based top-k queries have not been reported in the literature.

In this paper, we propose a new indexing framework for processing distance-based top-k queries. This framework exploits the ideas behind R-tree [5] and skyline algorithm [2], and thus called *SKY R-tree*. Notice that R-tree is a very efficient index for supporting spatial search while skyline algorithms are very effective for pruning the search space to find the best objects (i.e., skyline points) in terms of non-spatial preference. Thus, SKY R-tree aims to prune the search space by exploring the location and non-spatial attributes. Each node of SKY R-tree records a summary of location information and skyline points of objects in the sub-tree rooted at the node. Moreover, to avoid unnecessarily increasing the number of skyline points, each node employs *Abstract skyline* [11] to limit the number of skyline points stored. Utilizing the location information to estimate the spatial distance of candidate objects to a query point, and the skyline points to estimate the values of their non-spatial attributes, the proposed query processing algorithm is able to efficiently answer the distance-based top-k query.

[2] In this example, smaller values of non-spatial attributes are better.

To incorporate a similarity measure of non-spatial attributes, we also propose a variant of SKY R-tree, called *S2KY R-tree*. While SKY R-tree mainly considers location information in its construct, S2KY R-tree considers both the location information and non-spatial attributes, and thus is able to process distance-based top-k queries efficiently.

The main contributions of this paper are summarized as follows.

- We propose a new indexing technique which incorporates R-tree and skyline to support distance-based top-k queries.
- We propose an efficient algorithm for processing distance-based top-k queries.
- We demonstrate, through extensive experimentation, that our proposal performs very well in terms of both the I/O costs and CPU time.

The remainder of this paper is organized as follows. Section 2 describes the problem formulation. Section 3 presents the proposed index structure. Section 4 presents the enhanced approach. Section 5 summarizes the results obtained in the simulation experiments. Section 6 introduces related work. Section 7 summarizes the paper.

2 Preliminaries

2.1 Problem Formulation

Given an object dataset O in which each object o has location information $o.loc$ and non-spatial attribute $o.att$. We assume $o.loc$ is in a two dimensional geographical space composed of latitude and longitude, and $o.att$ is a d-dimensional vector where $o.att_i \in [0,1](i = 1, \cdots, d)$. We further assume that smaller values of these non-spatial attributes, e.g., price, are preferable, without loss of generality. Each score of object is calculated based on a query q which is represented by a query point $q.loc$ and a query weight $q.w$, where $q.w$ is a vector of weights such that $q.w_i \geq 0(i = 1, \cdots, d)$ and $\sum_{i=1}^{d} q.w_i = 1$. Accordingly, the ranking score of an object o is calculated by the following equation.

$$score(q,o) = \alpha \frac{D(q.loc, o.loc)}{maxD} + (1-\alpha)\frac{\sum_{i=1}^{d} q.w_i \cdot o.att_i}{maxA} \quad (1)$$

where $\alpha \in [0,1]$ is a parameter for balancing spatial proximity and non-spatial attributes, $D(q.loc, o.loc)$ is the Euclidian distance between q and o, and $maxD$ and $maxA$ are maximal distance and dissimilarity for normalization, respectively. Based on the above scoring function, the distance-based top-k query is defined as follows.

Definition 1 (Distance-Based Top-k Query). *Given a query q and the number of objects of interest k, the result of the distanced-based top-k query q includes a set of objects $TOP_k(q)$ such that $TOP_k(q) \subset O$, $|TOP_k(q)| = k$ and $\forall o_i, o_j : o_i \in TOP_k(q), o_j \in O - TOP_k(q)$, it holds that $score(q, o_i) \leq score(q, o_j)$.*

Notice that, in this paper, if some objects have the same kth score, the result retrieves one (or some) of the objects that meet the requirements of the $TOP_k(q)$.

Example: Recall the example in Figure 1. Let the query be q ($q.loc$=$\{0.5, 0.5\}$, $q.w$=$\{0.5, 0.5\}$), α be 0.5, and $k = 2$. By calculating the score of each object by Eq. (1), the top-1 and top-2 objects are o_7 and o_4, respectively.

2.2 Background

Here, we introduce Skyline and Max aR-tree which inspire our proposal of integrating R-tree and Skyline.

Skyline. In distance-based top-k queries, the scores of objects factor in both the location and non-spatial attributes. When focusing on only the non-spatial attributes, we can find the best object from skyline set because the top-k query is defined by linearly weighting the non-spatial attributes.

In the following we define the skyline set and discuss its relation to top-k queries.

Definition 2 (Skyline). *An object $o_i \in O$ is said to dominate another object $o_j \in O$, if for each attribute att_x $o_i.att_x \leq o_j.att_x$ and at least one attribute att_y $o_i.att_y < o_j.att_y$. The skyline is a set of points $SKY \subseteq O$ which are not dominated by any other points. The points in SKY are called skyline points.*

Observation 1. *The top-1 object obtained for a query that employs an increasingly monotone function on the non-spatial attributes belongs to the skyline set [12].*

From the skyline points, we can find the best objects (without considering the factor of location). Thus, skyline is useful to prune the search space from the aspect of non-spatial attributes.

Example: Consider the right graph in Figure 2 which plots the non-spatial attributes of the objects in Figure 1 in a two-dimensional space. As o_4 is dominated by o_7, o_4 is not a skyline point. In this figure, skyline points are o_1, o_2, o_6, o_7, and o_8.

Max aR-tree. *Max aR-tree* proposed in [7] can be utilized for distance-based top-k queries. In Max aR-tree, each node has a summary of location information and non-spatial attributes of descendant nodes. First, as location information, each node records a Minimum Bounding Rectangle (*MBR*) which is a rectangle contains all objects in the sub-tree rooted at the node. Additionally, each node in Max aR-tree records the best non-spatial attribute of each dimension in the sub-tree rooted at the node. Max aR-tree can prune the search space by non-spatial attribute, but it is not efficient because the best non-spatial attribute recorded in a node is quite different from an actual best score of object.

Example: Figure 3 illustrates a Max aR-tree indexing for the nine objects in Figure 1. Notice that, while Figure 2 shows the MBR recoded in each node and

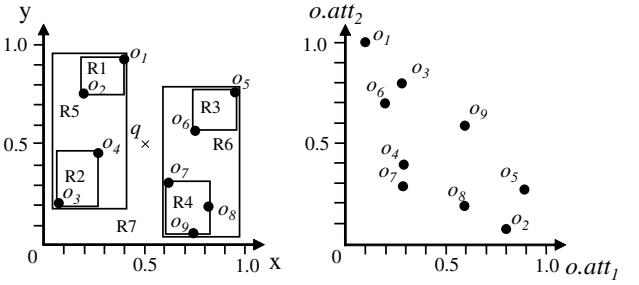


Fig. 2. Minimum bounding rectangles and non-spatial attributes

non-spatial attributes of each object, Table 1 shows the best values recorded in an internal node and non-spatial attributes of objects recorded in a leaf node. In this example, a leaf node R1 records (o_1:0.1, 1.0) and (o_2:0.8, 0.1), and an internal node R5 records (R1:0.1, 0.1) and (R2:0.3, 0.4). From R5's view, R1 seems to record the object with the best non-spatial attribute, however, the actual non-spatial attributes in the objects recorded in R1 are quite different.

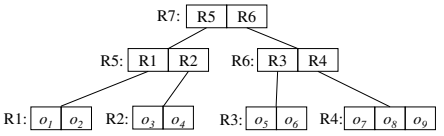


Fig. 3. Index structure in corresponding R-tree

Table 1. Best values which are recorded in each node in Max aR-tree

Node	Best value
R7	(R5:0.1,0.1), (R6:0.2,0.2)
R6	(R3:0.2,0.3), (R4:0.6,0.2)
R5	(R1:0.1,0.1), (R2:0.3,0.4)
R4	(o_7:0.3,0.3), (o_8:0.6,0.2), (o_9:0.6,0.6)
R3	(o_5:0.9,0.3), (o_6:0.2,0.7)
R2	(o_3:0.3,0.8), (o_4:0.3,0.4)
R1	(o_1:0.1,1.0), (o_2:0.8,0.1)

3 Hybrid Indexing for R-tree and Skyline

We present a new indexing framework that exploits the strengths of R-tree and skyline, called *SKY R-tree*. Moreover, we develop an algorithm for processing distance-based top-k queries.

3.1 SKY R-tree

SKY R-tree is essentially an R-tree, where each index node records skyline points of all objects in the sub-tree rooted at the node.

The design of SKY R-tree aims at pruning the search space by exploring non-spatial attribute. Ultimately, each node records all the non-spatial attributes of objects in a sub-tree rooted at the node in R-tree. To reduce the I/O cost in

R-tree, the number of child nodes is limited by a size of information recorded in one index node. Hence, if each node contains too large a volume of information, the efficiency decreases. Due to this constraint, it is important to select the information needed for efficiently pruning the search space. Our design considers two goals: (1) to guarantee an exact result, and (2) to represent a large amount of information in a compact form. In SKY R-tree, we choose skyline points because they elegantly meet these two goals.

When processing distance-based top-k queries, skyline points are very helpful to process the non-spatial attributes of candidate objects. However, the number of skyline points depends on the distribution of non-spatial attributes. If there are a large number of skyline points, the size of some nodes may become very large, resulting in deterioration of the performance. Therefore, SKY R-tree reduces the number of stored skyline points, while guaranteeing the correctness of query processing (and thus the result). If the number of skyline points exceeds a preset maximum number of skyline point $|\overline{SKY}|$, the skyline points are aggregated by *Abstract skyline* [11], which picks $|\overline{SKY}|$ *representative* skyline points from the entire skyline points.

In SKY R-tree, each leaf node contains a number of entries of the form *(o, o.loc, o.att)*. Meanwhile, an internal (non-leaf) node contains a number of entries of the form *(np, rectangle, sp)*, where *np* is a pointer to one of its child node, *rectangle* is the MBR, and *sp* is the skyline points of the child nodes.

Example: Figure 3 illustrates the SKY R-tree for the nine objects in Figure 1. Note that, while the index is structured as a MAX aR-tree, the information associated with the tree nodes is different. Let the maximum number of skyline points be 2. Table 2 shows the skyline points associated with an internal node as well as non-spatial attributes associated with objects in a leaf node. For example, R5 stores 3 skyline points (i.e., <0.1,1.0>, <0.8,0.1> for R1 and <0.3,0.4> for R2). As R7 can record only 2 skyline points for R5, we merge <0.1,1.0> and <0.3,0.4> into <0.1,0.4> and records it along with <0.8,0.1> for R5.

Table 2. Skyline points recorded in each node

Node	Skyline point
R7	(R5:<0.1,0.4>,<0.8,0.1>), (R6:<0.2,0.7>,<0.3,0.2>)
R6	(R3:<0.9,0.3>,<0.2,0.7>), (R4:<0.3,0.3>,<0.6,0.2>)
R5	(R1:<0.1,1.0>,<0.8,0.1>), (R2:<0.3,0.4>)
R4	(o_7:0.3,0.3), (o_8:0.6,0.2), (o_9:0.6,0.6)
R3	(o_5:0.9,0.3), (o_6:0.2,0.7)
R2	(o_3:0.3,0.8), (o_4:0.3,0.4)
R1	(o_1:0.1,1.0), (o_2:0.8,0.1)

We next present how to construct SKY R-tree. Similar to the R-tree algorithm [5], it repeats the Insert function in Algorithm 1, which uses a ChooseLeaf to select a leaf node with the smallest increase in MBR of the node in order to insert an object into the node as an entry. If the insertion exceeds the maximum node capacity, a Split is incurred to divide the node into two MBRs distant from each other.

Algorithm 1. Insert algorithm

```
1: N ← ChooseLeaf(object)
2: Add object to node N.
3: if exceed maximum number of child nodes of N then
4:    {N1, N2} ← N.Split()
5:    if N is root then
6:       Initialize new root NR and add N1 and N2 to NR
7:    else
8:       Ascend from N to root, and update MBR and skyline points
9:    end if
10: else if N ≠root then
11:    Update the MBR and skyline points of the ancestor node of N
12: end if
```

What makes SKY R-tree essentially different from the conventional R-tree is the update of the skyline points in ancestor node (line 11).

3.2 Distance-Based Top-k Query Processing

To process distance-based top-k queries, we exploit the best-first traversal. The best-first traversal algorithm searches the entry with the smallest score in a heap. The score for an individual object has been defined previously in Eq. (1). Therefore, we define the score for an index node N here as follows.

$$score(q, N) = \alpha \frac{MinD(q.loc, N.MBR)}{MaxD} + (1-\alpha)\text{argmin}_{\forall p \in N.sp} \sum_{i=1}^{d} q.w_i \cdot p.att_i \quad (2)$$

where $MinD(q.loc, N.MBR)$ denotes a minimum Euclidian distance between the MBR of N and $q.loc$. Eq. (2) provides a possible minimum score (i.e., lower bound) corresponding to the descendant objects of the node N.

In Algorithm 2, MINHEAP is a heap which contains index nodes and objects sorted in ascending order of their scores. Hence, if the first entry in MINHEAP is an object, it is the best object in MINHEAP (because the other objects definitely have larger scores than the first one). When the number of object in $TOP_k(q)$ exceeds k, the processing terminates (lines 6–7). Here, since objects and nodes in MINHEAP are sorted in ascending order, objects and nodes with larger scores than the top-k objects in MINHEAP are discarded (line 11). While the number of objects and nodes in MINHEAP becomes small, the computational overhead decreases because the sorting time becomes small.

Algorithm 2. Search algorithm

Input: k, q and index structure
Output: $TOP_k(q)$

```
 1: MINHEAP.insert(root, 0)
 2: while MINHEAP.size() ≠ 0 do
 3:    N ← MINHEAP.first()
 4:    if N is object then
 5:       TOP_k(q).insert(N)
 6:       if |TOP_k(q)| ≥ k then
 7:          return TOP_k(q)
 8:       end if
 9:    else
10:       for ∀n_i ∈ N.entry do
11:          if Number of objects with smaller score in MINHEAP than score(q, n_i) <
             (k − |TOP_k(q)|) then
12:             if N is a leaf node then
13:                MINEHEAP.insert(Object, score(q, n_i))
14:             else
15:                MINHEAP.insert(Node, score(q, n_i))
                   //If score is a same value, a smaller MBR is better.
16:             end if
17:          end if
18:       end for
19:    end if
20: end while
```

We prove that $TOP_k(q)$ is an exact result by the following theorem.

Theorem 1. *The score to an index node N is smaller than the scores of its descendant object o for any query q.*

Proof. The score factors in (i) location proximity and (ii) non-spatial attributes. First, the MBR of N includes all descendant objects, i.e., $\forall o \in$ descendant objects of N; $MinD(q.loc, N.MBR) \leq D(q.loc, o.loc)$. Second, the skyline points of N dominate or equal to all descendant objects, i.e., $\text{argmin}_{\forall p \in N.sp} \sum_{i=1}^{d} q.w_i \cdot p.att_i \leq \sum_{i=1}^{d} q.w_i \cdot o.att_i$. These inequalities imply that $score(q, N) \leq score(q, o)$. □

From this theorem, it follows that if the score of an object is smaller than the score of an index node, the score of the object is smaller than that of all descendant objects of the index node, i.e., the first object in MINHEAP is the best object.

4 Enhanced SKY R-tree

SKY R-tree is constructed based on locations of objects. However, the non-spatial attributes also play a role in the distance-based top-k queries. Thus, we argue that the locality of non-spatial attributes should also be considered in

construction of the index such that the cost of storage access can be reduced. Therefore, by considering a similarity measure of non-spatial attributes of objects, we propose a variant of SKY R-tree, called *S2KY R-tree* (similarity skyline R-tree).

4.1 S2KY R-tree

The S2KY R-tree inherits a similar idea from SKY R-tree, i.e., each index node records skyline points of its subtree. However, its clustering strategy is different. Thus, its ChooseLeaf and Split are different from that of SKY R-tree in Algorithm 1 by recording different child nodes at each internal node. Notice that SKY R-tree takes into account only the areas of MBRs under examination. On the other hand, S2KY R-tree takes into account both the areas of MBRs as well as the similarity of non-spatial attributes. To choose an appropriate insertion path for a new object, S2KY R-tree selects a child node as described below.

Let $E_1, \cdots, E_p$ be the entries in the current node, and let o be the object to be inserted. The corresponding R-tree calculates an increased area of MBR by the following equation.

$$AreaIncrease(o, E_x) = Area(o + E_x) - Area(E_x) \tag{3}$$

where $Area(x)$ is an area of the MBR including x, and E_x $(1 \leq x \leq p)$ is an entry (child node) in the current node. A small $AreaIncrease$ means the new object is close to the descendant objects in the child node.

On the other hand, S2KY R-tree also considers the similarity of non-spatial attributes which is defined as follows.

Definition 3 (Similarity measure of non-spatial attributes). *Similarity is a distance between non-spatial attributes of new object and skyline points in entry E.*

$$Similarity(o, E_x) = \min_{\forall p \in E_x.sp} dist(o.att, p) \tag{4}$$

where $dist(x, y)$ denotes the distance between non-spatial attribute of x and y.

Finally, we calculate $Increase$ by summing of $AreaIncrease$ and $Similarity$.

$$\begin{aligned} Increase(o, E_x) = \beta & \frac{AreaIncrease(o, E_x)}{MaxArea} \\ & + (1 - \beta) \frac{Similarity(o, E_x)}{MaxSim} \end{aligned} \tag{5}$$

where $MaxArea$ and $MaxSim$ are the size of entire area and the maximum similarity used for normalization, and β is a weighting parameter. The node with the smallest $Increase$ is chosen as the next node because a small $Increase$ means that the distance between E and o is a small (and the non-spatial attributes are similar). This calculation is repeated until progressing to a leaf node in Algorithm 3.

Algorithm 3. ChooseLeaf algorithm

```
1: N ← the root
2: while 1 do
3:    if N is a leaf node then
4:       return N
5:    end if
6:    N ← np in the entry in N with the smallest increase
7: end while
```

If β is 1 in Eq. (5), S2KY R-tree is the same as SKY R-tree. The best β is empirically determined by queries and distribution of non-spatial attribute.

Moreover, we define Split function and the difference between skyline sets in two child nodes by the following equation.

$$Sdist(SKY_A, SKY_B) = \frac{\sum_{p \in SKY_A} min_{\forall q \in SKY_B} dist(p,q)}{|SKY_A| + |SKY_B|} + \frac{\sum_{q \in SKY_B} min_{\forall p \in SKY_A} dist(q,p)}{|SKY_A| + |SKY_B|}. \quad (6)$$

This equation calculates the average minimum distance between skyline points. A small $Sdist(x, y)$ means that skyline sets x and y in the two nodes is closer to each other.

Algorithm 4. Split algorithm

```
1: for ∀ pair of E_i and E_j in entries do
2:    d_ij ← area(E_i + E_j) − area(E_i) − area(E_j).
3:    sim_ij ← Sdist(SKY_{E_i},SKY_{E_j});
4:    dif_ij = βd_ij + (1 − β)sim_ij
5: end for
6: Choose the pair with the largest dif value to be the first elements of the two group

7: while not all entries in N belong a group do
8:    if one group needs to include all remaining entries then
9:       Assign all remaining entries to it and break
10:   end if
11:   Calculate Increase for all remaining entries to two groups
12:   Choose the entry with the largest Increase and add it to the other group
13: end while
```

In Algorithm 4, the first two entries are selected as the first elements (lines 1–6). Other entries are assigned to either groups. To avoid assigning an entry with a large *Increase* to a group, the entry with the largest *Increase* is assigned to the other group in order (each entry has two *Increase*s due to two groups) (lines 11–12). Moreover, to avoid a bias of the number of child nodes in two groups, the minimum number of child nodes are determined in advance (basically a half of the maximum number of child nodes) (lines 8–9).

4.2 Query Processing

We use Algorithm 2 for processing of distance-based top-k queries on S2KY R-tree. In S2KY R-tree, each leaf node are more likely to record similar objects. Since k objects with better scores are inserted to MINHEAP earlier than SKY R-tree, the number of objects and nodes in MINHEAP becomes smaller. As a result, the computational overhead may be reduced.

5 Performance Evaluation

5.1 Simulation Model

We evaluate SKY R-tree and S2KY R-tree to validate our idea in processing the distance-based top-k query, by using Max aR-tree as the baseline for comparison.

Datasets. We use real location sets for objects, where the real location set is a set of points of interests located in Tokyo (10km $\times$ 10km) extracted from Foursquare. The dataset includes 45,129 objects. As the location set extracted from Foursquare has no non-spatial attribute, we added synthetic attributes to those points of interests. We use 2 types of synthetic datasets for non-spatial attributes: uniform and anti-correlated.

Setup. All index structures are in memory, and we assume the page size is 4KB. The size of an index node is $(16 + 4d \cdot |\overline{SKY}|)$B, i.e., each node has $\lceil 4,096/(16 + 4d \cdot |\overline{SKY}|) \rceil$ child nodes (In Max aR-tree, the size of an index is the same when $|\overline{SKY}| = 1$). For SKY R-tree and S2KY R-tree, the default values of $|\overline{SKY}|$ and β are set at 5 and 0.8, respectively. All algorithms implemented in c++, and experimented on a server with Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz with 8.00 GB RAM.

For each experiment, we generate 100 queries with randomly generated locations and weights. We use the average measures of "the number of I/O" (a number of visited nodes) and "CPU time" (query processing time) as the performance metrics. Table 3 shows parameters and values used in our experiments (the values in bold are the default values).

5.2 Experimental Result

To evaluate the performance of algorithms for the distance-based top-k queries, we vary two query parameters, k and α, as well as the dimension of non-spatial attributes, d, with 2 types of synthetic datasets for non-spatial attributes.

Uniform Non-spatial Attribute
Varying k. With k increasing, the search space increases. Figures 4a and 4d show the result in the uniform non-spatial attribute. We can see that SKY R-tree and S2KY R-tree outperform Max aR-tree for all values of k in terms of both the number of I/O and CPU time. From this result, we validate our idea of using skyline points to efficiently prune the search space. SKY R-tree works better than

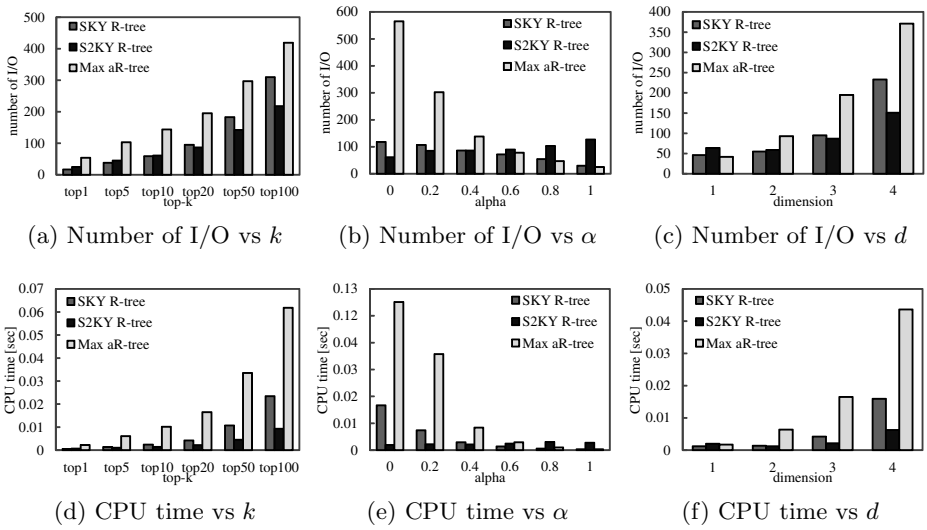
Table 3. Setting

Parameter	Value
Requested # of object k	1, 5, 10, **20**, 50, 100
Alpha	0, 0.2, **0.3**, 0.4, ,0.6, 0.8, 1.0
Dimension d	1, 2, **3** , 4
Attribute set	uniform, anti-correlated

S2KY R-tree in terms of the number of I/O for a small value of k, while S2KY R-tree works better for a large value of k. The MBR of an index node in SKY R-tree is smaller than that in S2KY R-tree, and thus the search space becomes smaller. In S2KY R-tree, as k increases, the probability for the top-k objects to have the same ancestor nodes increases, resulting in decrease in the number of I/O. Moreover, S2KY R-tree significantly outperforms SKY R-tree in terms of CPU time. Since S2KY R-tree likely finds objects with good score at one traversal, the number of nodes and objects in MINHEAP decreases, resulting in decrease of CPU time.

Varying α. Parameter α in Eq. (1) represents importance of location for a user. With α increasing, the location proximity contributes more than the non-spatial attributes to the ranking score. Figures 4b and 4e show the result in the uniform non-spatial attribute. When α is small, the performance S2KY R-tree is better than the other two methods. When α is large, the performance is worse than the others because S2KY R-tree takes into account the similarity measure of non-spatial attributes. In SKY R-tree and Max aR-tree, although the search space can be pruned by non-spatial attributes, the MBR of each node likely increases. Hence, with α increases, the number of I/O gradually increases. SKY R-tree outperforms Max aR-tree for all values of α because SKY R-tree can prune the search space more efficiently than Max aR-tree.

Varying d. With the number of dimensions increasing, the diversity of non-spatial attributes increases. Due to the curse of dimensionality, it becomes difficult to prune the search space by non-spatial attributes. Figures 4c and 4f show the result in the uniform non-spatial attribute. Our proposal outperforms Max aR-tree for all values of d, except for 1. Note that Max aR-tree works better than our proposal in the one dimension space because default setting for the number of skyline points is 5, but there is only one skyline point in one dimensional space. As a result, our setting wastes some redundant information in this case. With d increasing, a similarity of non-spatial attributes efficiently works, resulting in increase in the performance of S2KY R-tree more than SKY R-tree.



(a) Number of I/O vs k (b) Number of I/O vs α (c) Number of I/O vs d

(d) CPU time vs k (e) CPU time vs α (f) CPU time vs d

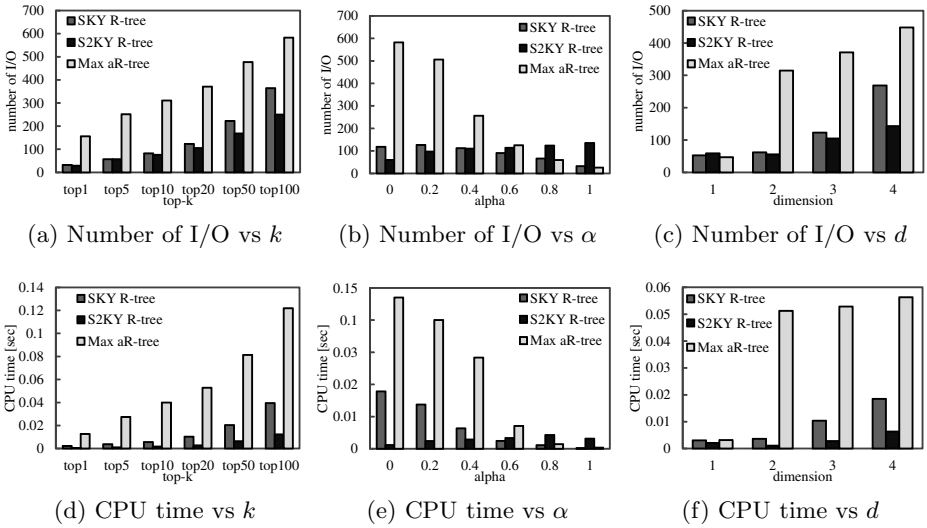
Fig. 4. Results in uniform non-spatial attribute

Anti-correlated Attribute

Varying k. Figures 5a and 5d show the result with the anti-correlated non-spatial attributes. The performance in S2KY R-tree does not change much compared with the uniform attribute set. On the other hand, SKY R-tree and Max aR-tree on this dataset work less efficiently than the uniform attribute set. In the anti-correlated set, SKY R-tree and Max aR-tree cannot efficiently prune the search space upon the non-spatial attributes, while S2KY R-tree keeps high efficiency by taking into account the similarity of non-spatial attributes. Specially, in Max aR-tree, almost all nodes record the non-spatial attributes near the best one (i.e., non-spatial attributes of each dimension are almost zero), and thus the performance is significantly worse. From this result, we can see that S2KY R-tree is robust for any non-spatial attributes by considering the similarity of non-spatial attributes.

Varying α. Figures 5b and 5e show the result with anti-correlated non-spatial attribute. The trend of results in our proposal is similar to Figure 4b. On the other hand, when α is small, Max aR-tree works worse because pruning the search space by non-spatial attribute does not work well in the anti-correlated attributes.

Varying d. Figures 5c and 5f show the result in the anti-correlated non-spatial attribute. S2KY R-tree outperforms the other methods for all values of d except for 1. The variation of non-spatial attributes in anti-correlated is smaller than the uniform, and thus the similarity of non-spatial attributes works better.



(a) Number of I/O vs k (b) Number of I/O vs α (c) Number of I/O vs d

(d) CPU time vs k (e) CPU time vs α (f) CPU time vs d

Fig. 5. Results in anti-correlated non-spatial attribute

6 Related Work

Top-k Query. Conventional top-k query processing has been extensively studied [6]. Fagin et al. [4] propose a class of algorithms known as threshold algorithms for computing the top-k query from multiple lists. Threshold algorithm accesses tuples from the database in a certain order, and maintains an upper bound as the maximum score for the unseen objects. If the upper bound is lower than the score of top-k objects, the algorithm terminates. Various threshold algorithms [1] have been proposed to improve some of their limitation and to study in other application areas. Tao et al. [9] propose a branch-and-bound algorithm for top-k queries. This branch-and-bound algorithm, based on R-tree [5], visits nodes which record the potential object with the best score.

These algorithms can be used for processing distance-based top-k queries, but they are inefficient. For example, by finding the top-1 object based on non-spatial attributes, and then adding the score of location proximity, we can repeatedly find the top-k objects. However, these algorithms probably search redundant objects whose locations are further from a query points.

Nearest Neighbor Query. k nearest neighbor (kNN) query processing in spatial databases, which retrieves the k objects closest to a query points, is also related to our work. kNN query processing proposed by Roussopoulos et al. [8] maintains potential nearest neighbors in a queue based on R-tree [5], and traverses the tree to find the best objects. For example, it finds the nearest neighbor, and then adds the score of non-spatial attributes. However, kNN queries processing based on R-tree are worse than a MAX aR-tree approach.

7 Conclusion

In this paper, we propose a new index structure for distance-based top-k query. The index structure, named *SKY R-tree*, incorporates R-tree and skyline to prune the search space by exploring both the spatial proximity and non-spatial attributes. Accordingly, we propose an algorithm for processing distance-based top-k queries. The index structure and processing algorithm, forming a new framework, is very efficient to retrieve the best objects. Moreover, we consider the similarity measure of non-spatial attributes of objects in its construction in a new index, named *S2KY R-tree*. Through extensive experimentation, we demonstrate that our proposal efficiently processes distance-based top-k queries in terms of both the I/O costs and CPU time. The difference between performance of SKY R-tree and S2KY R-tree depends on the situations under different importance of location proximity and the distribution of locations of objects and non-spatial attributes.

This work can be extended in several directions for future work. First, it would cooperate with keyword search, e.g., the "Japanese" restaurant close to a query point with high rating. Second, continuous distance-based top-k query processing for mobile users would be interesting. Third, in this paper we calculate the distance between objects and a query point in Euclidian distance, while a road-based distance is also an important challenge.

Acknowledgment. This research is partially supported by the Grant-in-Aid for Scientific Research (S)(21220002), (B)(24300037), and JSPS Fellows (24-293) of the Ministry of Education, Culture, Sports, Science and Technology, Japan.

References

[1] Akbarinia, R., Pacitti, E., Valduriez, P.: Best position algorithms for top-k queries. In: VLDB, pp. 495–506 (2007)
[2] Borzsony, S., Kossmann, D., Stocker, K.: The skyline operator. In: ICDE, pp. 421–430 (2001)
[3] Cong, G., Jensen, C.S., Wu, D.: Efficient retrieval of the top-k most relevant spatial web objects. VLDB Journal 2(1), 337–348 (2009)
[4] Fagin, R., Lotem, A., Naor, M.: Optimal aggregation algorithms for middleware. Journal of Computer and System Sciences 66(4), 614–656 (2003)
[5] Guttman, A.: R-trees: a dynamic index structure for spatial searching. SIGMOD 14, 47–57 (1984)
[6] Ilyas, I.F., Beskales, G., Soliman, M.A.: A survey of top-k query processing techniques in relational database systems. ACM CSUR 40(4), 11 (2008)
[7] Lazaridis, I., Mehrotra, S.: Progressive approximate aggregate queries with a multi-resolution tree structure. ACM SIGMOD Record 30(2), 401–412 (2001)
[8] Roussopoulos, N., Kelley, S., Vincent, F.: Nearest neighbor queries. ACM SIGMOD Record 24(2), 71–79 (1995)
[9] Tao, Y., Hristidis, V., Papadias, D., Papakonstantinou, Y.: Branch-and-bound processing of ranked queries. Information Systems 32(3), 424–445 (2007)

[10] Vlachou, A., Doulkeridis, C., Nørvåg, K.: Monitoring reverse top-k queries over mobile devices. In: MobiDE, pp. 17–24 (2011)
[11] Vlachou, A., Doulkeridis, C., Nørvåg, K.: Distributed top-k query processing by exploiting skyline summaries. IEEE Tran. on Distributed and Parallel Databases 30(3-4), 239–271 (2012)
[12] Vlachou, A., Doulkeridis, C., Nørvåg, K., Vazirgiannis, M.: On efficient top-k query processing in highly distributed environments. In: ACM SIGMOD, pp. 753–764 (2008)