DART: An Efficient Method for Direction-aware Bichromatic Reverse k Nearest Neighbor Queries

Kyoung-Won Lee[1], Dong-Wan Choi[2], and Chin-Wan Chung[1,2]

[1]Division of Web Science Technology, Korea Advanced Institute of Science & Technology, Korea
[2]Department of Computer Science, Korea Advanced Institute of Science & Technology, Korea
{kyoungwon.lee,dongwan}@islab.kaist.ac.kr,chungcw@kaist.edu

Abstract. This paper presents a novel type of queries in spatial databases, called *the direction-aware bichromatic reverse k nearest neighbor*($DBRkNN$) queries, which extend the bichromatic reverse nearest neighbor queries. Given two disjoint sets, P and S, of spatial objects, and a query object q in S, the DBRkNN query returns a subset P' of P such that k nearest neighbors of each object in P' include q and each object in P' has a direction toward q within a pre-defined distance. We formally define the DBRkNN query, and then propose an efficient algorithm, called *DART*, for processing the DBRkNN query. Our method utilizes a grid-based index to cluster the spatial objects, and the B^+-tree to index the direction angle. We adopt a filter-refinement framework that is widely used in many algorithms for reverse nearest neighbor queries. In the filtering step, DART eliminates all the objects that are away from the query object more than the pre-defined distance, or have an invalid direction angle. In the refinement step, remaining objects are verified whether the query object is actually one of the k nearest neighbors of them. From extensive experiments, we show that DART outperforms an R-tree-based naive algorithm in both indexing time and query processing time.

Keywords: reverse nearest neighbor, direction-aware, query optimization

1 Introduction

Recently, with the rapid dissemination of mobile devices and location-based services(LBSs), various applications have started utilizing spatial databases for mobile users. *Bichromatic reverse nearest neighbor*($BRNN$) queries extended from *reverse nearest neighbor*(RNN) queries are one of the most popular and important queries for spatio-temporal information, and widely used in many applications. For example, in the case of mobile advertising, an advertiser can promote a product to specifically targeted customers who are close to the advertiser based on each customer's location by searching BRNNs of the advertiser. Many researches addressed that one of the future challenges of location-based services is

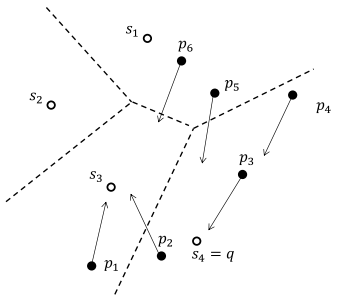


Fig. 1. An example of BRNN query with a direction constraint

personalization [5,12,15], which provides more customized services, based on the user's implicit behaviour and preferences, and explicitly given details. In order to achieve personalization in LBSs, considering only location is not sufficient to retrieve more accurate results in terms of the user's intention.

In that respect, the direction is another important feature that represents user's intention as there exist extensive researches that consider the direction to predict moving object's future location [17]. Each mobile user can have a certain direction with respect to his/her movement or sight, and the direction can be easily obtained by a mobile device with GPS and a compass sensor [18]. However, there are only a few researches that considers a direction-aware environment, and existing studies only focus on user-centric query processing, not objective-centric query processing.

Considering the above, BRNN queries without the direction constraints can be ineffective in many applications to find targeted users in the sense that users looking(or moving) in the opposite direction are less influenced by the query objects even if they are close to the query object. For example, there are many customers in a marketplace, and they are moving around and looking for some products they need. In this situation, a restaurant manager may want to find potential customers who have an intention to enter the marketplace, and hang around the restaurant, because the manager wants to reduce the advertisement budget and does not want to be regarded as a spammer to customers. There are also other kinds of applications where a direction property needs to be considered such as providing a battle strategy to a moving military squad during the war. In these applications, the direction as well as the location are important properties to obtain more accurate results in terms of the targets' intention.

Fig. 1 shows an example of the BRNN query with a direction constraint. Consider a set $P = \{p_1, p_2, p_3, p_4, p_5, p_6\}$ of customers and a set $S = \{s_1, s_2, s_3, s_4\}$ of advertisers. Given a querying advertiser s_4, the usual BRNN query returns p_2, p_3 and p_4 since their closest advertiser is q (i.e., s_4). However, the customers whose directions (represented by arrows) are not toward q do not need to be considered because they are not effective advertising targets. Thus, although customer p_2 has q as its nearest advertiser, p_2 should be discarded from the final result in the direction-aware environment since the direction of p_2 is toward s_3, not q.

Furthermore, in order to maximize the effectiveness of advertising, it is better to consider the distance. If we adjust a maximum distance on p_4, it also can be discarded depending on the maximum distance, even if their nearest advertiser is q. Therefore, only p_3 can be an answer for the BRNN query with the direction constraint.

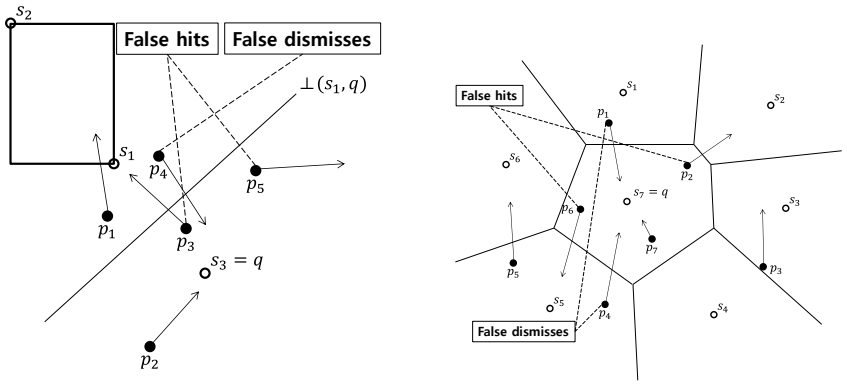
There have been extensive algorithms studied for processing RNN queries [1,10,19,22] and BRNN queries [14,21,25,26], based on various effective pruning techniques using objects' locations. However, the straightforward adaptation of these algorithms are inefficient to solve the problem of finding BRNNs with the direction constraint. This is because each object has an arbitrary direction, which does not have any correlation with its location.

In this paper, we present a novel type of queries, called *direction-aware bichromatic reverse k nearest neighbor queries*(*DBRkNN*), in spatial databases, which extends the previous BRNN query by considering the direction as well as the location. Moreover, we propose an efficient algorithm, called *DART*, for our DBRkNN queries to overcome the difficulties of pruning in a direction-aware environment. DART attempts to minimize pre-processing time by using only a grid-based index to access the set of spatially clustered objects and the B^+-tree to index objects' directions. In common with many previous studies, we follow a filter-refinement framework. In specific, in the filtering step, DART returns a set of candidates, each of which has q as one of its k nearest neighbors and a direction toward q within a pre-defined distance, while the refinement step removes false hits from the set of candidates.

The contributions of this paper are as follows:

- We propose a novel type of query, the direction-aware bichromatic reverse k nearest neighbor (DBRkNN) query, which is an interesting variant of the bichromatic reverse nearest neighbor query. The DBRkNN query is useful in many applications which require to process a large amount of spatial objects with arbitrary directions.
- We propose an efficient algorithm, namely DART, to process DBRkNN queries specially focusing on a direction-aware pruning technique. To effectively prune unnecessary objects, DART uses simple index structures and yet significantly reduces the pre-processing time.
- We experimentally evaluate the proposed algorithm by using synthetic datasets. Experimental results show that our proposed algorithm is on the average 6.5 times faster for the indexing time, and 6.4 times faster for the query processing time than an R-tree-based naive algorithm.

The rest of the paper is organized as follows. Section 2 surveys related work. Section 3 presents the formal definition of the DBRNN query. The proposed algorithm for the DBRNN query is explained in Section 4. Section 5 experimentally evaluates the proposed algorithm. Finally, Section 6 concludes the paper with some directions for future work.



(a) An example of TPL algorithm (b) An example of Voronoi diagram

Fig. 2. An example of the false hits/false dismisses of the previous works

2 Related Work

We first examine existing studies [1,2,4,9–11,14,19–24,26] about the RNN query, which has received considerable attention due to its importance and effectiveness in many applications. The first algorithm for processing the RNN query was proposed by Korn, F. et al [10]. However, this algorithm requires to index all data points, and to pre-compute their nearest neighbors which is inefficient in dynamic database environments. Stanoi et al. [19] proposed "60-degree-pruning", which maintains only an index tree without any pre-processing structure. They divide the space around the query point into six equal regions having 60° at the query point, and the answers are retrieved by selecting a candidate point from each region. The TPL algorithm was proposed by Tao et al [22], which utilizes the perpendicular bisector between the query point and each point to maximize the pruning area.

The above algorithms for RNN queries, however, are inefficient to process the DBRkNN query since they do not consider the direction constraint for the query processing. For example, Fig. 2(a) shows the false hits/false dismisses of the TPL algorithm for the DBRkNN query. There is a bisector between the query object q (i.e., s_3) and object s_1, so p_2, p_3, and p_5 are selected as candidates since they reside in the half-plane containing q. Although the object p_3 and p_5 are the BRNNs of q, these are not the DBRNNs of q because their directions are not toward q. Moreover, the pruned object p_4 that is located in the opposite half-plane to q can be an answer in the DBRNN query, because its direction is toward q across the bisector.

The traditional RNN queries have been further branched out into the bichromatic RNN (BRNN) query, which is the closest to our DBRNN query. Given two disjoint sets, P and S, of points, and a query point which is one of the points in S, the BRNN query retrieves a set of points in P that have q as their nearest neighbor. There are some researches [9, 14, 20, 25] on the BRNN query,

and all their solutions are basically focusing on finding the Voronoi polygon that contains the query point by using a Voronoi diagram.

However, the above methods for the BRNN query are not efficient in our environment. For example, Fig. 2(b) shows an example of using Voronoi diagram to solve BRNN query. There are seven Voronoi polygons each of which is generated by an object in S. In the case of p_2 and p_6, they are the BRNN of q (i.e., s_7), but the directions of them are toward the opposite side of q, which makes them to be false hits/false dismisses. Similarly, although p_4 and p_1 are not the BRNN of q, they are the DBRNN of q because their directions are toward q.

For other types of RNN queries, there has been many researches for processing the *continuous RNN*(*CRNN*) [2–4, 6, 9, 21] query and the *stream RNN* [11] query. The goal of each type of queries is basically to find the RNN with regard to the query object in a specific environment. However, the solutions for these queries are neither applicable nor relevant to the DBRkNN query due to the properties of the query.

By focusing on the influence of obstacles on the visibility of objects, there are works on a different type of RNN queries [7, 16, 27]. Gao et al. [7] first introduced the *visible reverse nearest neighbor* (*VRNN*) search, which considers the visibility and the obstacle that significantly affect the result of RNN queries. However, the visibility is defined only for the query object to verify objects that are not influenced by the presence of obstacles while the direction in the DBRNN query is defined for each object to represent its movement or sight. Furthermore, we also adjust the maximum distance to give a flexibility on the spatial environment.

Recently, Li et al. [13] proposed the *direction-aware spatial keyword search*, called *DESKS*, that finds the k nearest neighbors satisfying both keyword and direction constraints to the query. They assumed that the direction is given and addressed that the existing methods on the spatial keyword search are inefficient to solve the spatial keyword search with a direction constraint. However there is a big difference between this work and ours in the sense that we focus on RkNN queries (not kNN), and every object has a direction in our environment while only the query object has a direction in DESKS.

3 Problem Formulation

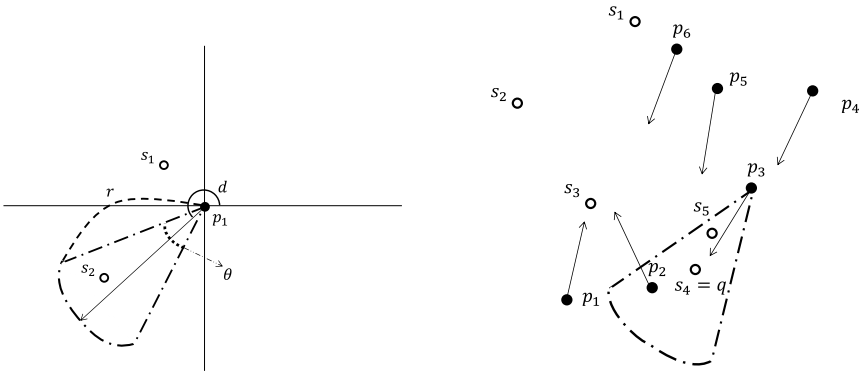
In this section, we formally define the DBRNN query along with the DBRkNN query. Table 1 summarizes the notations frequently used.

3.1 Problem Definition

We consider two disjoint sets, P and S, of spatial objects, and a query object q in S. Each object in P, called a *customer* object, includes its location and a direction which is represented by a counterclockwise angle from the positive x-axis (i.e., the direction of 3 o'clock is 0°), and has a fan-shaped region, called a *valid area*, based on its *direction's angle*. On the other hand, objects in S, called

Table 1. The notation of the DBRkNN

Symbol	Description
$P = \{p_1, ..., p_n\}$	the set of customer objects with directions
$S = \{s_1, ..., s_m\}$	the set of advertiser objects
p	a customer object with a direction in P
s	an advertiser object in S
q	the query object selected from advertiser objects in S
r	the maximum distance
d	the direction angle ($0^\circ \sim 359^\circ$)
θ	the valid angle range



(a) An illustration of the DBRNN query

(b) An illustration of the DBRkNN query ($k = 2$)

Fig. 3. An illustration of the DBRNN query and the DBRkNN query

advertiser objects, have only locations, and one of the advertiser objects can be a query object. We first define the notion "valid area" which is an important concept for selecting candidates and verifying answers as follows:

Definition 1 (Valid Area). *Let p denote an object in P. Then the valid area of p is represented by a fan-shaped region, which has the following properties:*

- *A valid area consists of angle θ and radius r, both of which are pre-defined by a system.*
- *θ is a viewing angle, and r is the maximum distance to discard an object the distance of which is greater than r.*

Now, based on Definition 1, we define the DBRNN query as follows:

Definition 2 (Direction-aware Bichromatic Reverse Nearest Neighbor Query). *Given two disjoint sets, P and S, the direction-aware bichromatic reverse nearest neighbor query retrieves the subset P' of P such that each object in P' has $q \in S$ as its nearest neighbor, and contains q in its valid area.*

Fig. 3(a) illustrates the basic concept of the DBRNN query. The object p_1 has a valid area based on the maximum distance r, the direction's angle d and the valid angle range θ. If the query is invoked on s_2, p_1 is the DBRNN of s_2 even though s_1 is closer than s_2 since only s_2 is located within the valid area of p_1. Similar to Definition 2, the DBRkNN query can be defined as follows:

Definition 3 (Direction-aware Bichromatic Reverse k Nearest Neighbor Query). *Given two disjoint sets, P and S, the direction-aware bichromatic reverse k nearest neighbor query retrieves the subset P' of such P that each object in P' has $q \in S$ as one of its k nearest neighbors, and contains q in its valid area.*

Fig. 3(b) shows an illustration of the DBRkNN query. Let us assume that k is 2, and the query is invoked on s_4. If the value of k is 1, there is no answer for the query. However, in this case, although s_5 is the nearest neighbor of p_3 as well as a DBRkNN of q, because q is the second-nearest neighbor of p_3.

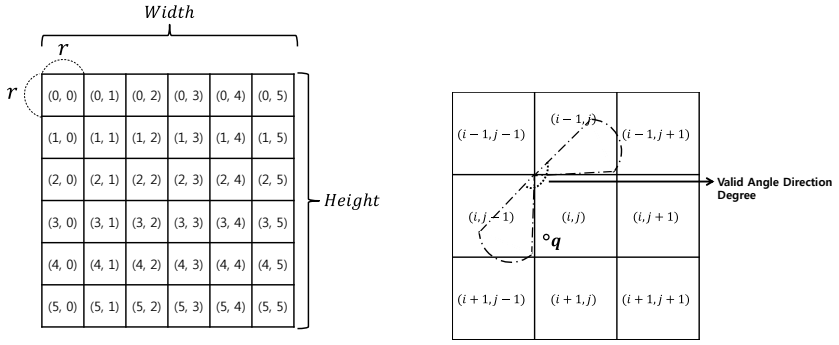
Problem Statement In this paper, our goal is to find an efficient method that gives the set of exact answers for the DBRNN query and the DBRkNN query. Specifically, we focus on minimizing both the indexing time and the query time.

4 The DART Algorithm

In this section, we present our proposed algorithm, called DART, that solves DBRNN queries. First, we overview our proposed method, and then explain the details of DART.

4.1 Overview

Essentially, our solution is based on a grid-based index to access the spatially clustered objects and the B$^+$-tree to index the direction's angles. We adopt a filter-refinement framework that is widely used in many algorithms for RNN queries. In the filtering step, DART eliminates all the objects that are more than the maximum distance away from the query object or have an invalid direction's angle. After that, in the refinement step, the remaining objects are verified to check whether the query object is actually the nearest neighbor of each object. The important features of DART are explained as follows:



(a) An example of the grid-based space partitioning

(b) An example of the Valid Direction Angle Degree

Fig. 4. The key elements for DART

The grid-based space partitioning The whole space is divided into a grid of the equal-sized cells that are represented by rectangles of $r \times r$ size (recall that r is the maximum distance). The number of rows and columns depends on the *width* and *height* of the space. Each cell has a unique id number that represents its location. Fig. 4(a) shows an example of our space partitioning scheme using grid cells. For each cell, we not only maintain two lists of objects (advertiser object and customer object) that are located in the area of the cell but also index the direction's angle of each customer object by using the B$^+$-tree. Note that this space partitioning takes just linear time while an R-tree takes at least $O(n \log n)$ time complexity for indexing n spatial objects [8].

Direction Angle Index The directions' angles are indexed by the B$^+$-tree to reduce unnecessary checks for the objects that are toward the wrong direction. In this structure, there are at most 360 keys which represent degrees of the directions' angles. For each key, we maintain a list of the objects that have the same direction angle degree as the key value. Similar to the grid-based space partitioning, the construction of this B$^+$-tree can be done in linear time, since each insertion requires only $O(\log 360)$ time (i.e., a constant time).

Valid Direction Angle Range Each grid cell has a static valid direction angle range (hereafter called "valid angle range") that guarantees, if the direction angle of an object is not within the valid angle range, the object cannot have an appropriate direction toward the query object. Fig. 4(b) shows an example of the valid angle range. When the query is posed, we first figure out which grid cell contains the query object, and then retrieve neighboring cells around the grid cell that has the query object. For each neighboring cell, we define the valid angle range accordingly. As we discussed in Section 3, we use counterclockwise angles; the 3 o'clock position is 0° and the 6 o'clock position is 270°.

Let us first consider the valid angle range of cell $(i-1, j-1)$. In an extreme case, an object in P in cell $(i-1, j-1)$ can be located at the bottom right corner of the cell and the query object can be located at the top right corner or the bottom left corner of cell (i, j). In this case, the valid angle range of an object in P should cover the top or left boundary of cell (i, j) to have the query object within its valid area. Therefore, the valid angle range of cell $(i-1, j-1)$ should be $(0°, \frac{\theta}{2}°)$ and $(270° - \frac{\theta}{2}°, 360°]$ as depicted in Fig. 4(b). The valid angle range of other three corner cells (i.e., $(i-1, j+1), (i+1, j-1)$, and $(i+1, j+1)$) are defined in a similar way. On the other hand, the cells on the cross line (i.e., $(i-1, j), (i, j-1), (i, j+1), (i+1, j)$) are defined in a different manner due to the positional characteristics. For example, in an extreme case of an object in P in cell $(i-1, j)$ can be located at the bottom left or bottom right corner of the cell and the query object can be located at the opposite side of the object in the cell (i, j). In this case, the valid angle range of an object in P should cover the top boundary of cell (i, j) to have the query object within its valid area. Therefore, the valid angle range of cell $(i-1, j)$ should be $(0°, \frac{\theta}{2}°)$ and $(180° - \frac{\theta}{2}°, 360°]$. Similar to the corner cells, the other cells on the cross line have similar valid angle ranges. The specific ranges of the valid angle ranges are shown in Table 2.

Table 2. The Valid Angle Range of each cell

Cell No.	**Valid Angle Degree**
$(i-1, j-1)$	$(0°, \frac{\theta}{2}°)$ and $(270° - \frac{\theta}{2}°, 360°]$
$(i-1, j)$	$(0°, \frac{\theta}{2}°)$ and $(180° - \frac{\theta}{2}°, 360°]$
$(i-1, j+1)$	$(180° - \frac{\theta}{2}°, 270° + \frac{\theta}{2}°)$
$(i, j-1)$	$(0°, 90° + \frac{\theta}{2}°)$ and $(270° - \frac{\theta}{2}°, 360°]$
(i, j)	$(0°, 360°]$
$(i, j+1)$	$(90° - \frac{\theta}{2}°, 270° + \frac{\theta}{2}°)$
$(i+1, j-1)$	$(0°, 90° + \frac{\theta}{2}°)$ and $(180° - \frac{\theta}{2}°, 360°]$
$(i+1, j)$	$(0°, 180° + \frac{\theta}{2}°)$ and $(360° - \frac{\theta}{2}°, 360°]$
$(i+1, j+1)$	$(90° - \frac{\theta}{2}°, 180° + \frac{\theta}{2}°)$

4.2 DART for DBRNN Query Processing

Based on the above key features, DART follows a two-step framework, where a set of candidate objects are returned and then false hits are verified to retrieve the exact solution. Our algorithm does not index the exact locations of the objects, but use a grid-based structure to access the set of spatially clustered objects efficiently. Moreover, our method inserts the object's direction's angle degree into the B^+-tree to maintain the object's direction's angle.

Algorithm 1: The construction of the basic structure

```
   Input: Sets of objects P and S
   Output: A set G of grid cell's list and the B+-tree
 1 G ← ∅;
 2 foreach obj in P ∪ S do
 3 |   cellId ← Assign(obj.x, obj.y) ;
 4 |   list ← lists[cellId].;
 5 |   list.add(obj);
 6 |   G.add(list);
 7 |   if obj ∈ P then
 8 |   |   d ← obj.d;
 9 |   |   Btree ← Btrees[cellId];
10 |   |   Btree.insert(d, obj);
11 |   end
12 end
13 return G;
```

Index Construction Step Algorithm 1 shows the process of constructing the basic structures. When an object is inserted, the assignment algorithm determines which grid cell contains the object (Line 3). In addition, the method just adds the object into the proper list and stores the list for the corresponding cell (Lines 4-6). For the two types of objects, we maintain two lists of objects separately. In addition, DART also maintains a B$^+$-tree to index direction's angle degree for each cell (Lines 8-10). As mentioned earlier, the entire construction process can performed in linear time.

Filtering Step In the filtering step, DART eliminates unnecessary objects by considering the maximum distance and the valid angle range based on the location and angle of the object. The overall algorithm flow is shown in Algorithm 2.

First, the method gets the grid cell number that contains the query object by using *assign* function. In this function, it is easy to retrieve the neighboring cells by using the *width* and *height* of the space (Lines 2-3). Because the size of each cell is determined by the maximum distance, we do not have to consider other cells except for the neighboring cells. By doing this, we can prune numerous objects whose distances from the query object are larger than the maximum distance.

Next, for each cell, DART selects the candidate set by processing the range search on the B$^+$-trees on directions' angles of objects located in the cell(Lines 5-7). Note that this range search requires only a constant time since there are at most 360 keys. As we discussed the valid angle range in Section 4.1, we can easily find the objects whose directions' angles are within the valid angle ranges of the corresponding cells (See Table 2). Before we put an object into the candidate set, we double check the actual distance and the angle degree between the query

Algorithm 2: The filtering step of DART

```
   Input: The query object q
   Output: A set of candidates
 1 candidate ← ∅;
 2 CellId ← Assign(query.x, query.y);
 3 neighbor[ ] ←getNeighbor(CellId);
 4 for i ← 0 to size of neighbor[ ] do
 5 |   Btree ← neighbor[i].Btree;
 6 |   range ← neighbor[i].ValidAngleRange;
 7 |   list[] ← Btree.rangequery(range);
 8 |   for j ← 0 to size of list[ ] do
 9 |   |   angle ← getAngle(q, list[j]);
10 |   |   if getDistance(list[j], q) ≤ r AND |(angle − list[j].d)| ≤ θ/2 then
11 |   |   |   candidate.add(list[j]);
12 |   |   end
13 |   end
14 end
15 return candidate
```

object and the object (Lines 8-13). The reason for doing this step is to guarantee that the candidate set contains only objects whose valid area cover the query object. If the distance between the two objects is within the maximum distance and the difference of the two directions' angles (angle degree between two objects and the object's direction angle degree) is less than $\frac{\theta}{2}$, then we finally add the object to the candidate set.

Refinement Step After the termination of the filtering step, we have a candidate set that contains all the objects whose valid area contain the query object. In the refinement step, we verify that the actual nearest neighbor of each candidate object is the query object. Algorithm 3 shows the flow of the refinement step. Basically, the method confirms the answer set by checking the nearest neighbor of each candidate. For candidate object p, if there is an advertiser object s_i closer than the query object, it is possible that s_i is the nearest neighbor of p and is within the valid area of p. To determine this, for all the advertiser objects in S that are contained in neighboring cells, the method calculates the distance between the candidate object and each advertiser object (Line 7). Moreover, if there is an advertiser object closer than the query object, we check the actual angle degree (Lines 7-13). Similar to the above procedure, if the difference is smaller than half of θ, it means that the direction's angle is also facing the object s_i, and the nearest neighbor of the candidate object is not the query object (Lines 9-12). Otherwise, the candidate object can be an answer of the DBRNN query.

Algorithm 3: The refinement step of DART for the DBRNN query

```
   Input: A candidate set C
   Output: a set of answers Answer
 1 neighbor[] ← getNeighborGrid();
 2 list ←getObjectS(neighbor[]);
 3 Answer ← C;
 4 for i ← 0 to size of C do
 5 |   distance ← getDistance(q, C[i]);
 6 |   for j ← 0 to size of list[] do
 7 |   |   if distance > getDistance(C[i], list[j]) then
 8 |   |   |   angleS ← getAngle(C[i], list[j]);
 9 |   |   |   if |(angleS − C[i].d)| ≤ θ/2 then
10 |   |   |   |   Answer.remove(C[i]);
11 |   |   |   |   break;
12 |   |   |   end
13 |   |   end
14 |   end
15 end
16 return Answer
```

4.3 DART for the DBRkNN Query Processing

In this section, we extend the algorithm for DBRNN queries to process DBRkNN queries for an arbitrary value of k, which means the query result should be all the customer objects that have q within k nearest neighbors (k is a positive integer, typically small). For processing DBRkNN queries, although the arbitrary value k is added, the overall flow is almost the same. The filtering step does not need to be changed, because we prune the unnecessary objects only considering the distance and angle constraints. In the refinement step, DART should be slightly modified so that k advertiser objects can be checked when finding advertiser objects closer than the query object (Lines 10-11 in Algorithm 3).

5 Experiments

In this section, we evaluate the performance of our proposed algorithm for the DBRNN query and the DBRkNN query by using four synthetic datasets. In particular, we generate synthetic datasets for both spatial object sets, P and S, under the uniform distribution. We set the size of the dataset for spatial objects in P to be from 10,000 to 10,000,000 and that in S to be $|P|/100$ on the 2-dimensional 1,000 × 1,000 euclidean space. For experimental parameters, we vary the valid angle range, the maximum distance, and the cardinality of the dataset. The values of parameters are presented in Table 3.

The experiment investigates the index time and query time for varying values of parameters such as the valid angle range, the maximum distance, and the cardinality. For comparison, we also implement a naive method that utilizes an

Parameter	The range of values
Valid angle range	30 - 90 degree (60 degree by default)
Maximum distance	50 - 200 (100 by default)
Cardinality of P	10,000 - 10,000,000 (1,000,000 by default)

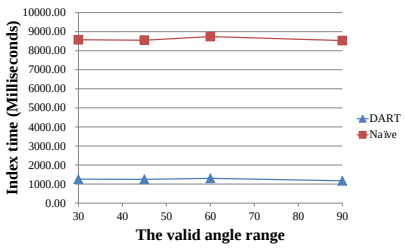
Table 3. The values of parameters

R-tree on the objects' locations because the existing methods for RNN queries and BRNN queries cannot guarantee the exact solution for DBRNN queries, which implies that the methods are not applicable to DBRNN queries and the DBRkNN queries (See Fig. 2). Similar to the proposed algorithm, the naive method also prunes the objects outside of the circular range that has a radius equal to a maximum distance. After that, this method conducts the same procedure as the refinement step to remove false hits. The only difference from DART is that the direction pruning is not performed in the filtering step. All algorithms are implemented in Java, and the experiments are conducted on a PC equipped with Intel Core i7 CPU 3.4GHz and 16GB memory.

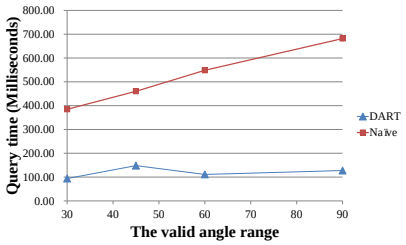
5.1 Experimental Results of DBRNN Query

Fig. 5 shows the performance of DART and the naive method when processing DBRNN queries with varying values of experimental parameters. Fig. 5(a), 5(c), and 5(e) represent the index time of each experiment, and Fig. 5(b), 5(d), and 5(f) represent the query time. For all results on the index and query processing, DART shows a superior performance compared to the naive method.

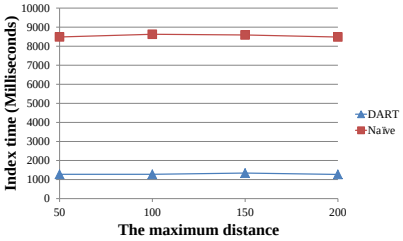
First, we conduct an experiment with varying the valid angle range varying from 30 degree to 90 degree. According to Fig. 5(a), the grid-based clustering and the B^+-tree indexing on the direction's angles do not need heavy indexing time while the R-tree based indexing is time consuming. We can observe that the grid-based clustering takes less time than the R-tree to maintain the object's location. Moreover, the direction angle indexing time is not a big issue in the whole pre-processing step because we have at most 360 angle degrees so that there are at most 360 keys as we mentioned earlier. On the other hand, Fig. 5(b) shows an increasing gap between the query time of DART and that of the naive method. The reason that the naive method shows an increasing curve as the valid angle range increases is due to the total candidates of answer objects. For instance, DART filters irrelevant objects with its valid angle range, and then conducts refinement on a candidate set. However, the naive method just filters objects which have a longer distance than the maximum distance by conducting range search on the R-tree, and checks for the direction's angle for each object. In this step, the naive method generates more candidates as the valid angle range gets wider, and hence the number of total candidates increases.



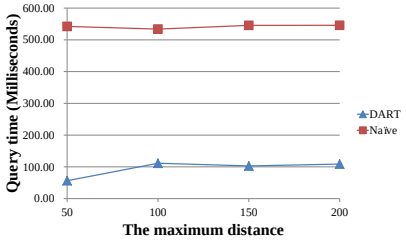
(a) Index time for varying valid angle range



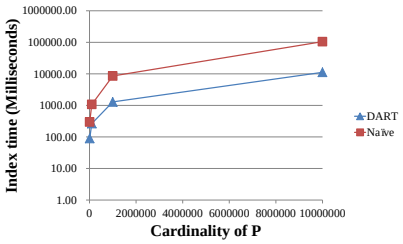
(b) Query time for varying valid angle range



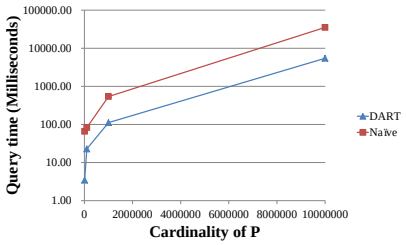
(c) Index time for varying maximum distance



(d) Query time for varying maximum distance



(e) Index time for varying cardinality



(f) Query time for varying cardinality

Fig. 5. Experimental Results of DBRNN Query

According to Fig. 5(c), indexing time is almost similar to the result of the valid angle range, and Fig. 5(d) indicates that the query processing time shows a steady gap between DART and the naive method. There is a little increasing line for DART for the maximum distance from 50 to 100, because the number of total candidates is quite small for the data3 dataset.

In Fig. 5(e) and Fig. 5(f), we conduct experiments with varying the cardinality by using all the datasets. Both index time and query time show similar increasing curves as the cardinality becomes bigger, however, the difference of the performance is upto 10 times between DART and the naive method. In our observation, DART can handle bigger sized datasets more efficiently than the naive method, even though the dataset reaches 10 millions of objects.

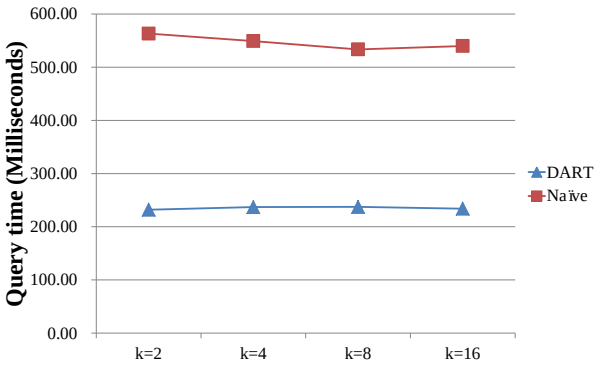


Fig. 6. Experimental Results of DBRkNN Query

5.2 Experimental Results of DBRkNN Query

Fig. 6 shows the performance of DART and the naive method for the DBRkNN queries. For the arbitrary k value, we start from $k = 2$ and exponentially increase k until 16. The experimental results show that the query times for the cases are almost uniformly distributed. In our observation, this is due to the maximum distance and the direction constraint. Only a small computation is increased because the constraints limit the boundary for the DBRkNN search. From this result, we can claim that our proposed algorithm is also much more efficient for processing the DBRkNN query than the naive method.

5.3 Summary

In summary, we have shown through extensive experiments that DART outperforms the naive method in both indexing time and query processing time. We conducted several experiments by changing the values of parameters, namely the valid angle area, the maximum distance, and the cardinality to show the effect of those parameters on the performances. The results indicate that DART can handle more than 10 million objects within a minute. Therefore, DART is suitable for a snapshot query with at most one minute time interval to secure index time and query time. In addition, although DART approximately prunes irrelevant objects by using a grid-based space partitioning(while the naive method prunes certain objects whose distance are longer than the maximum distance), its direction angle pruning technique makes up the time of double checking for the maximum distance.

6 Conclusion

In this work, we presented a novel type of the RNN query that has a direction constraint, and proposed an efficient query processing algorithm called DART. Our algorithm utilizes the grid-based object clustering and the direction angle

indexing with the B^+-tree to improve both index time and query time. We also experimentally showed that the proposed algorithm outperforms the naive algorithm that utilizes the R-tree based range query pruning.

An interesting direction for future work is to extend our work to process the larger size of data more efficiently and to develop an algorithm that efficiently prunes unnecessary objects in S. Moreover, for the DBRkNN query, we plan to work on not only query optimization, but also finding more new characteristics of the direction constraint that can speed up algorithms.

Acknowledgments. This work was supported in part by WCU (World Class University) program under the National Research Foundation of Korea funded by the Ministry of Education, Science and Technology of Korea (No. R31-30007), and in part by the National Research Foundation of Korea grant funded by the Korean government (MSIP) (No. NRF-2009-0081365).

References

1. Achtert, E., Böhm, C., Kröger, P., Kunath, P., Pryakhin, A., Renz, M.: Efficient reverse k-nearest neighbor search in arbitrary metric spaces. In: Proceedings of the 2006 ACM SIGMOD international conference on Management of data. SIGMOD '06, New York, NY, USA, ACM (2006) 515–526
2. Benetis, R., Jensen, S., Karciauskas, G., Saltenis, S.: Nearest and reverse nearest neighbor queries for moving objects. The VLDB Journal **15**(3) (September 2006) 229–249
3. Cheema, M.A., Zhang, W., Lin, X., Zhang, Y.: Efficiently processing snapshot and continuous reverse k nearest neighbors queries. The VLDB Journal **21**(5) (October 2012) 703–728
4. Cheema, M.A., Zhang, W., Lin, X., Zhang, Y., Li, X.: Continuous reverse k nearest neighbors queries in euclidean space and in spatial networks. The VLDB Journal **21**(1) (February 2012) 69–95
5. Dhar, S., Varshney, U.: Challenges and business models for mobile location-based services and advertising. Commun. ACM **54**(5) (May 2011) 121–128
6. Emrich, T., Kriegel, H.P., Kröger, P., Renz, M., Xu, N., Züfle, A.: Reverse k-nearest neighbor monitoring on mobile objects. In: Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems. GIS '10, New York, NY, USA, ACM (2010) 494–497
7. Gao, Y., Zheng, B., Chen, G., Lee, W.C., Lee, K.C.K., Li, Q.: Visible reverse k-nearest neighbor query processing in spatial databases. IEEE Trans. on Knowl. and Data Eng. **21**(9) (September 2009) 1314–1327
8. Guttman, A.: R-trees: a dynamic index structure for spatial searching. In: Proceedings of the 1984 ACM SIGMOD international conference on Management of data. SIGMOD '84, New York, NY, USA, ACM (1984) 47–57
9. Kang, J.M., Mokbel, M.F., Shekhar, S., Xia, T., Zhang, D.: Continuous evaluation of monochromatic and bichromatic reverse nearest neighbors. In: In ICDE. (2007)
10. Korn, F., Muthukrishnan, S.: Influence sets based on reverse nearest neighbor queries. In: Proceedings of the 2000 ACM SIGMOD international conference on Management of data. SIGMOD '00, New York, NY, USA, ACM (2000) 201–212

11. Korn, F., Muthukrishnan, S., Srivastava, D.: Reverse nearest neighbor aggregates over data streams. In: Proceedings of the 28th international conference on Very Large Data Bases. VLDB '02, VLDB Endowment (2002) 814–825
12. Krumm, J.: Ubiquitous advertising: The killer application for the 21st century. Pervasive Computing, IEEE **10**(1) (jan.-march 2011) 66 –73
13. Li, G., Feng, J., Xu, J.: Desks: Direction-aware spatial keyword search. In Kementsietsidis, A., Salles, M.A.V., eds.: ICDE, IEEE Computer Society (2012) 474–485
14. Lian, X., Chen, L.: Monochromatic and bichromatic reverse skyline search over uncertain databases. In: Proceedings of the 2008 ACM SIGMOD international conference on Management of data. SIGMOD '08, New York, NY, USA, ACM (2008) 213–226
15. Mokbel, M.F., Levandoski, J.J.: Toward context and preference-aware location-based services. In: Proceedings of the Eighth ACM International Workshop on Data Engineering for Wireless and Mobile Access. MobiDE '09, New York, NY, USA, ACM (2009) 25–32
16. Nutanong, S., Tanin, E., Zhang, R.: Incremental evaluation of visible nearest neighbor queries. IEEE Trans. on Knowl. and Data Eng. **22**(5) (May 2010) 665–681
17. Qiao, S., Tang, C., Jin, H., Long, T., Dai, S., Ku, Y., Chau, M.: Putmode: prediction of uncertain trajectories in moving objects databases. Applied Intelligence **33** (2010) 370–386
18. Qin, C., Bao, X., Roy Choudhury, R., Nelakuditi, S.: Tagsense: a smartphone-based approach to automatic image tagging. In: Proceedings of the 9th international conference on Mobile systems, applications, and services. MobiSys '11, New York, NY, USA, ACM (2011) 1–14
19. Stanoi, I., Agrawal, D., Abbadi, A.E.: Reverse nearest neighbor queries for dynamic databases. In: In ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery. (2000) 44–53
20. Stanoi, I., Riedewald, M., Agrawal, D., Abbadi, A.E.: Discovery of influence sets in frequently updated databases. In: Proceedings of the 27th International Conference on Very Large Data Bases. VLDB '01, San Francisco, CA, USA, Morgan Kaufmann Publishers Inc. (2001) 99–108
21. Taniar, D., Safar, M., Tran, Q.T., Rahayu, W., Park, J.H.: Spatial network rnn queries in gis. Comput. J. **54**(4) (April 2011) 617–627
22. Tao, Y., Papadias, D., Lian, X.: Reverse knn search in arbitrary dimensionality. In: Proceedings of the Thirtieth international conference on Very large data bases - Volume 30. VLDB '04, VLDB Endowment (2004) 744–755
23. Tao, Y., Papadias, D., Lian, X., Xiao, X.: Multidimensional reverse knn search. The VLDB Journal **16**(3) (July 2007) 293–316
24. Tao, Y., Yiu, M.L., Mamoulis, N.: Reverse nearest neighbor search in metric spaces. IEEE Trans. on Knowl. and Data Eng. **18**(9) (September 2006) 1239–1252
25. Tran, Q.T., Taniar, D., Safar, M.: Bichromatic reverse nearest-neighbor search in mobile systems. IEEE Systems Journal **4**(2) (2010) 230–242
26. Vlachou, A., Doulkeridis, C., Kotidis, Y., Norvag, K.: Monochromatic and bichromatic reverse top-k queries. IEEE Trans. on Knowl. and Data Eng. **23**(8) (August 2011) 1215–1229
27. Wang, Y., Gao, Y., Chen, L., Chen, G., Li, Q.: All-visible-k-nearest-neighbor queries. In: DEXA (2). (2012) 392–407