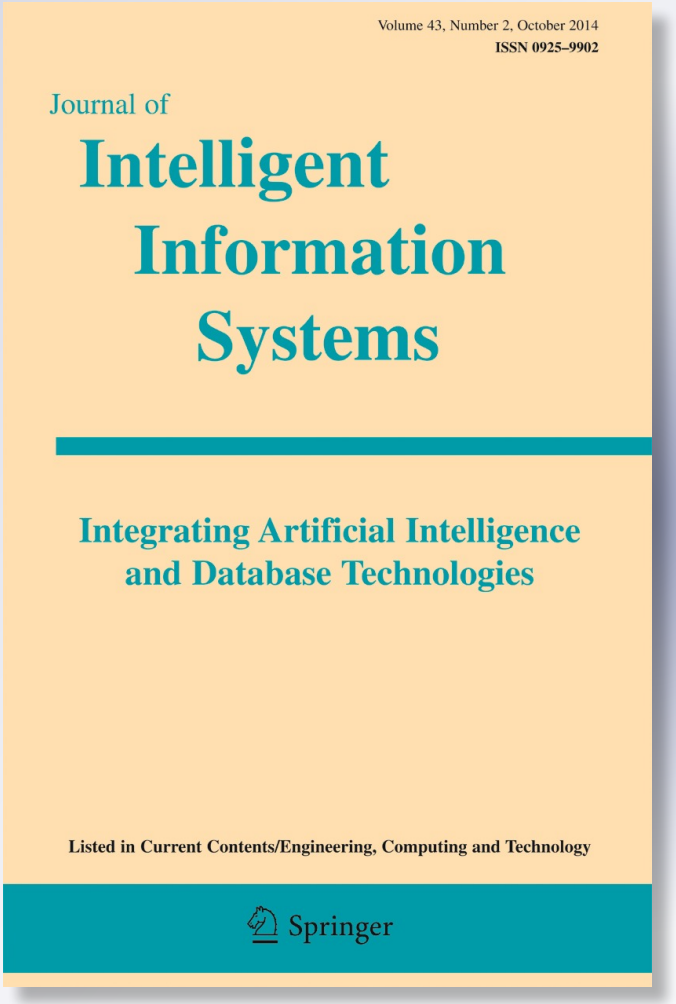
DART+: Direction-aware bichromatic reverse k nearest neighbor query processing in spatial databases

Kyoung-Won Lee, Dong-Wan Choi & Chin-Wan Chung

Journal of Intelligent Information Systems
Integrating Artificial Intelligence and Database Technologies

ISSN 0925-9902
Volume 43
Number 2

J Intell Inf Syst (2014) 43:349-377
DOI 10.1007/s10844-014-0326-3



Springer

J Intell Inf Syst (2014) 43:349–377
DOI 10.1007/s10844-014-0326-3

DART+: Direction-aware bichromatic reverse *k* nearest neighbor query processing in spatial databases

Kyoung-Won Lee · Dong-Wan Choi · Chin-Wan Chung

Received: 14 September 2013 / Revised: 27 May 2014 / Accepted: 28 May 2014 /
Published online: 20 June 2014

Abstract This article 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 a 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. As a major extension of DART, we also present an improved algorithm, called DART+, for DBRkNN queries. From extensive experiments with several datasets, we show that DART outperforms an R-tree-based naive algorithm in both indexing time and query processing time. In addition, our extension algorithm, DART+, also shows significantly better performance than DART.

Keywords Reverse nearest neighbor · Direction-aware · Query optimization

K-W. Lee
Division of Web Science and Technology, Korea Advanced Institute of Science and Technology, Daejeon, Korea
e-mail: kyoungwon.lee@islab.kaist.ac.kr

D-W. Choi
Department of Computer Science, Korea Advanced Institute of Science and Technology, Daejeon, Korea
e-mail: dongwan@islab.kaist.ac.kr

C-W. Chung (✉)
Division of Web Science and Technology and Department of Computer Science, Korea Advanced Institute of Science and Technology, Daejeon, Korea
e-mail: chungcw@kaist.edu

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 *personalization* (Dhar and Varshney 2011; Mokbel and Levandoski 2009; Krumm 2011), 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 (Qiao et al. 2010). 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 (Qin et al. 2011). 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.

Figure 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 (Achtert et al. 2006; Korn and Muthukrishnan 2000; Stanoi et al. 2000; Tao et al. 2004) and BRNN queries (Lian and Chen 2008; Taniar et al. 2011; Tran et al. 2010; Vlachou et al. 2011), based on various effective pruning techniques using objects' locations. However, the straightforward

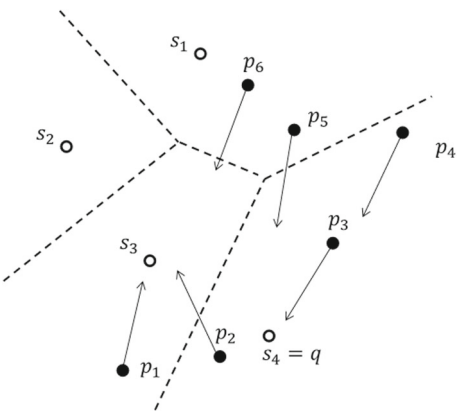


Fig. 1 An example of BRNN query with a direction constraint

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.

As a major extension of DART, this article also propose an improved algorithm, called *DART+*, for DBRkNN queries to reduce the number of conducting the refinement processes that must be performed by DART. DART+ attempts to minimize the refinement step to improve the previous algorithm by using a dynamic filtering with a cropping method. Using this method, we can significantly prune unnecessary objects in the refinement step.

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 propose an improved algorithm, namely DART+, to process DBRkNN queries specially focusing on an efficient refinement process by using the dynamic filtering. To conduct the dynamic filtering, DART+ uses a cropping method and significantly reduces the query processing time.

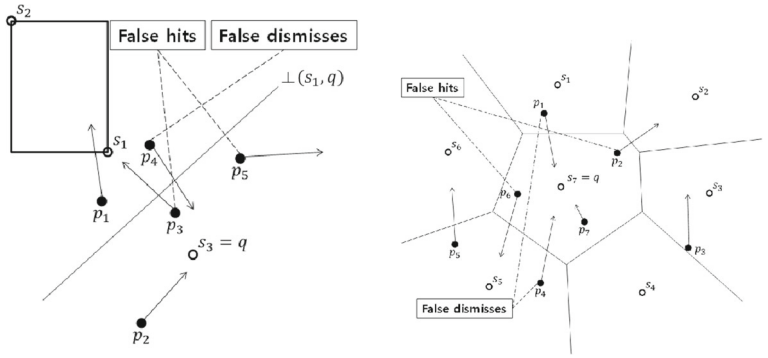
– We experimentally evaluate the proposed algorithm by using synthetic datasets and real datasets. Experimental results show that DART is on the average 6.5 times faster for the indexing time, and 6.4 times faster for the query processing time than an R-tree-based naive algorithm. Moreover, DART+ is on the average 3 times faster for the query processing time than DART.

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 naive algorithm for the DBRNN query is presented in Section 4, and one of proposed algorithms, called DART, for the DBRNN query is explained in Section 5. Section 6 explains a major extension of DART, called DART+, Section 7 experimentally evaluates the proposed algorithms. Finally, Section 8 concludes the paper with some directions for future work.

2 Related work

We first examine existing studies (Achtert et al. 2006; Benetis et al. 2006; Cheema et al. 2012; Kang et al. 2007; Korn and Muthukrishnan 2000; Korn et al. 2002; Lian and Chen 2008; Stanoi et al. 2000, 2001; Taniar et al. 2011; Tao et al. 2004, 2006, 2007; Vlachou et al. 2011) 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 and Muthukrishnan (2000). 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. (2000) 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. (2004), 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. 2a shows the false hits/false dismisses of the TPL algorithm for the DBRkNN



(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

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 (Tran et al. 2010; Kang et al. 2007; Lian and Chen 2008; Stanoi et al. 2001) 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. 2b 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*) query and the *stream RNN* 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. This is because the solutions for these queries use some techniques such as a bisector or a partition to construct several safe zones for pruning irrelevant objects, which do not consider the direction feature, and it is another challenge to modify those solutions to work also in a direction-aware environment.

By focusing on the influence of obstacles on the visibility of objects, there are works on a different type of RNN queries (Gao et al. 2009; Wang et al. 2012; Nutanong et al. 2010). Gao et al. (2009) 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. (2012) 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.

Finally, a preliminary version of this article was published in Lee et al. (2013). The authors proposed the direction-aware bichromatic reverse k nearest neighbor query, called DBRkNN, that finds the reverse k nearest neighbor satisfying the direction constraint to the query. They defined two types of spatial objects, one has an arbitrary direction and location information, and the other has only location information. One of the static objects

becomes the query. In addition, they also presented a naive algorithm and DART that solve the DBRkNN queries.

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 *directional angle*. On the other hand, objects in S, called *advertiser* objects, have only locations, and one of the advertiser objects can be a query object. Intuitively, the valid area of a customer object can be seen as an influential area that the object can affect the other type of objects during the query processing. We now formally define the valid area which is important 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.

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 directional angle ($0° \sim 359°$)
θ	the valid angle range

Figure 3a illustrates the basic concept of the DBRNN query. The object p_1 has a valid area based on the maximum distance r, the directional 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.

Figure 3b 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.

3.1.1 Problem statement

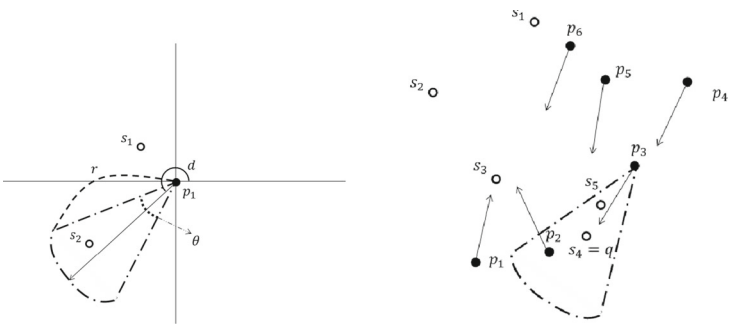
In this paper, our goal is to find an efficient method that gives the set of exact answers for the DBRNN query and the DBRkNN query. Specifically, we focus on minimizing both the indexing time and the query time.

4 The naive algorithm

In this section, we present a naive algorithm that solves DBRNN queries. First, we overview the algorithm, and then explain the details of the algorithm.

4.1 Overview

The naive algorithm is based on an R-tree index structure to access the spatial objects with its specific location information. Moreover, we adopt a filter-refinement framework to follow procedures of ordinary RNN query processing algorithms. In the filtering step, the



(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

algorithm excludes all the objects that are more than the maximum distance away from the query object. In addition, the algorithm also checks the directional angle to make sure the remaining objects guarantee a proper distance as well as a proper directional angle. In the refinement step, the remaining objects are verified to check whether the query object is considered as the nearest neighbor of each object. The important feature of the naive algorithm is explained as follows:

Algorithm 1 The filtering step of the naive algorithm

```
   Input: The query object q
   Output: A set of candidates
 1 candidate ← ∅;
 2 UserTree ← getUserTree();
 3 list[]← UserTree.rangeSearch(q, r);
 4 for j ← 0 to size of list[ ] do
 5 |   angle ← getAngle(q, list[j]);
 6 |   if |(angle − list[j].d)| ≤ θ/2 then
 7 |   |   candidate.add(list[j]);
 8 |   end
 9 end
10 return candidate
```

4.1.1 Spatial object indexing using an R-tree

Every spatial objects in a set P are indexed by an R-tree structure (Guttman 1984). The algorithm maintains the spatial object's specific location information to prune unnecessary objects that are out of the maximum distance. Using this index structure, the algorithm can achieve almost exact distance pruning with a range search.

4.2 A naive algorithm for DBRNN query processing

Based on the above key feature, the naive algorithm follows a filter-refinement framework to retrieve the exact answer. The algorithm indexes the specific location of the objects, but does not consider the directional angle. Therefore, although it does not maintain the directional angle of the objects, its one of advantages is conducting a nearly optimal distance pruning with an R-tree indexing structure.

4.2.1 Index construction step

In the index construction step, the algorithm inserts objects' coordinates into an R-tree. In this process, the algorithm inserts only objects in P, because all of objects in S need to be utilized in the refinement step to verify that a candidate object considers the query objects as its nearest neighbor or not.

4.2.2 Filtering step

In the filtering step, the naive algorithm eliminates unnecessary objects by considering the maximum distance based on the location of the object. The overall filtering process is shown in Algorithm 1.

Algorithm 2 The refinement step of the naive algorithm

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

First, the algorithm gets the user's location information from the R-tree structure. Using this structure, we can easily get nearest objects within the maximum distance by conducting a range search (Line 2-3). As shown in Fig. 4, the algorithm can effectively prune unnecessary objects with a nearly optimal distance range search. By doing this, we can ignore a lot of objects whose distance from the query object is larger than the maximum distance, and the pruning result is almost optimal.

Next, for each remaining object, the algorithm double checks the angle degree between the query object and the object (Line 6-8). The reason for doing this step is to make sure that the candidate set contains only objects whose valid area covers the query object. Therefore, if the difference of the two directional angles is less than $\frac{\theta}{2}$, then we finally add the object to the candidate set.

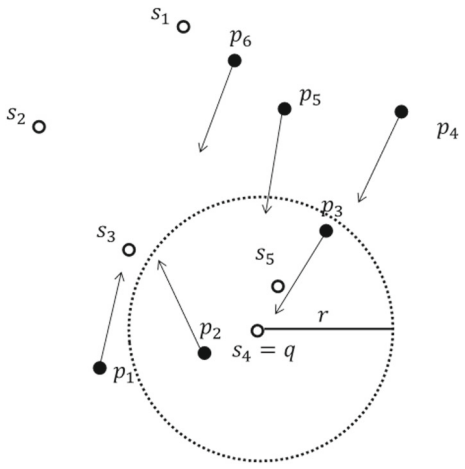


Fig. 4 A range query of the native algorithm

4.2.3 Refinement step

After the process of the filtering step, we obtain a candidate set that includes all the objects whose valid area contains the query object. In this step, we examine that the actual nearest neighbor of each candidate object is the query object. Algorithm 2 shows the flow of the refinement step. Basically, the method confirms the answer set by checking the nearest neighbor of each candidate. For a 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, the method calculates the distance between the candidate object and each advertiser object (Line 4). Moreover, if there is an advertiser object closer than the query object, we check the actual angle degree (Lines 7-12). Similar to the above procedure, if the difference is smaller than a half of θ, it means that the directional angle is also facing the object s_i, and the nearest neighbor of the candidate object is not the query object (Lines 8-11). Otherwise, the candidate object can be an answer of the DBRNN query.

4.3 A naive algorithm for DBRkNN query processing

In this section, we extend the algorithm for DBRNN queries to process DBRkNN queries for an arbitrary value 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 unnecessary objects only considering the distance and angle constraints. In the refinement step, the naive algorithm should be slightly modified so that k advertiser objects can be checked when finding advertiser objects closer than the query object (Lines 9-10 in Algorithm 2). In the for loop (Lines 5-13), we skip the removal process (Line 9) until the number of proper objects reaches $k - 1$, and once the number equals to k, then remove the candidate object.

5 The DART algorithm

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

5.1 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:

5.1.1 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. Figure 5a 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(Guttman 1984).

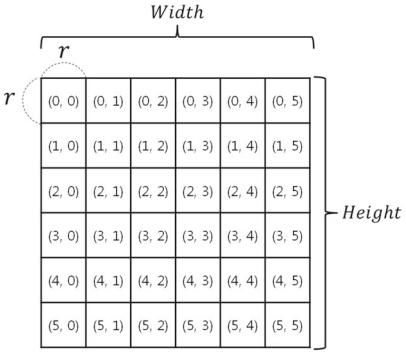
5.1.2 Direction angle index

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).

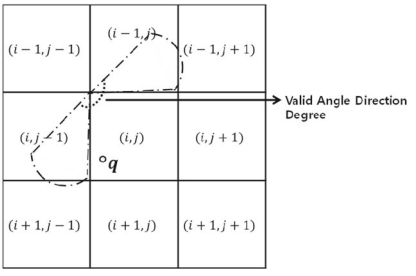
5.1.3 Valid direction angle range

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. Figure 5b 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



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



(b) An example of the Valid Direction Angle Degree

Fig. 5 The key elements for DART

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. 5. 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.

5.2 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.

5.2.1 Index construction step

Algorithm 3 shows the process of constructing the basic structures. When an object is inserted, the assignment algorithm determines which grid cell contains the object (Line 4). In addition, the method just adds the object into the proper list and stores the list for the corresponding cell (Lines 5-7). 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 9-11). As mentioned earlier, the entire construction process can performed in linear time.

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 $(360° - \frac{\theta}{2}°, 360°]$
$(i+1, j+1)$	$(90° - \frac{\theta}{2}°, 180° + \frac{\theta}{2}°)$

5.2.2 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 4.

Algorithm 3 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      obj ← object[i];
4      cellId ← Assign(obj.x, obj.y) ;
5      list ← lists[cellId].;
6      list.add(obj);
7      G.add(list);
8      if obj ∈ P then
9          d ← obj.d;
10         Btree ← Btrees[cellId];
11         Btree.insert(d, obj);
12     end
13 end
14 return G;
```

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 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.

5.2.3 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 5 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

Algorithm 4 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 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 6). Moreover, if there is an advertiser object closer than the query object, we check the actual angle degree (Lines 6-12). 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 8-11). Otherwise, the candidate object can be an answer of the DBRNN query.

5.3 DART for the DBR*k*NN query processing

In this section, we extend the algorithm for DBRNN queries to process DBR*k*NN queries for an arbitrary value k, which means the query result should be all the

Algorithm 5 The refinement step of DART for the DBRNN query

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

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 9-10 in Algorithm 5) in a way similar to the naive algorithm as explained in Section 4.3.

6 The DART+ Algorithm

In this section, we present our proposed algorithm, called DART+, that improves DART to solve DBRNN queries more efficiently. First, we overview our proposed method, and then explain the details of DART+ (Fig. 6).

6.1 Overview

Although DART effectively prunes unnecessary objects using the grid-based space partitioning, the directional angle index, and valid direction angle range, this algorithm does not handle objects in S. In other words, the refinement step is similar to that of the naive algorithm so that there is a possibility of improving the algorithm. Therefore, DART+ focuses on the refinement step to minimize the trials of the refinement process using a dynamic filtering for the objects in the set S, while the index construction step and the filtering step are the same as those of DART. The important features of DART+ are explained as follows:

6.1.1 A cropping method

As we mentioned earlier, the objective of DART+ is to improve the refinement step of DART. To achieve this, we conduct a dynamic filtering on the objects in S for each candidate object in P. Basic observation is that we can get the grid cells that overlap the valid area of a candidate object. This can be done by finding the minimum bounding rectangle(MBR) with representative points of the valid area. These representative points can be obtained by using the polar coordinate system, and the details will be explained in Section 6.1.2. Thus, we can get grid cells intersecting with the MBR. Figure 7a shows the first step of a cropping method. By considering the grid cells, we can discard unnecessary objects in other grid cells which do not affect the candidate object. To enhance the accuracy and efficiency of the dynamic filtering, we verify whether the object is within the MBR or not for the remaining

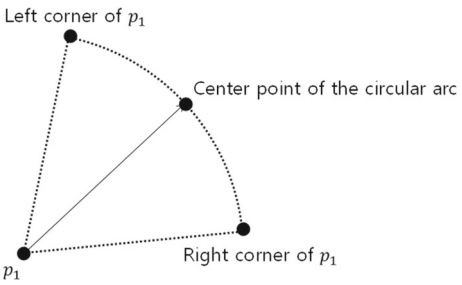
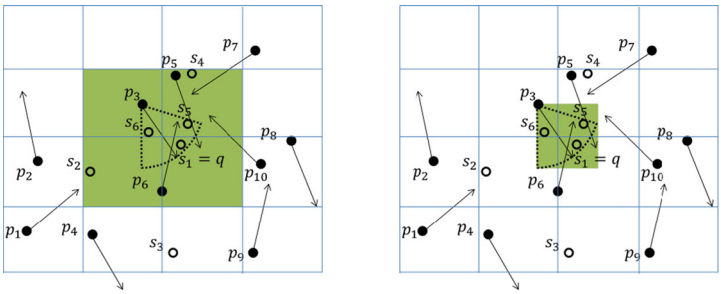


Fig. 6 Three points to conduct the dynamic filtering



(a) The first step of the cropping method

(b) The second step of the cropping method

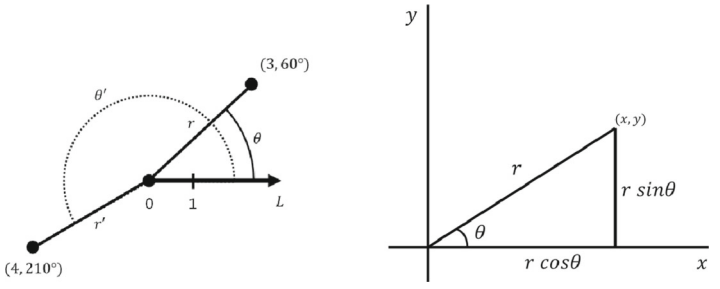
Fig. 7 An example of the cropping method

objects. If the object is within the MBR, it has the possibility of being the nearest neighbor of the candidate object.

6.1.2 The polar coordinate system

The most important task in the cropping method is determining corner points of each candidate object's valid area. To achieve this task, DART+ utilizes the polar coordinate system. The polar coordinate system is a two-dimensional coordinate system that can be obtained by the distance from the fixed point and an angle from a fixed direction. Figure 8a shows an example of the polar coordinate system. In DART+, each candidate object's position is a fixed point, called the pole, and the object's directional angle is θ.

However, we can not adapt this coordinate system directly, because we need to know the cartesian coordinates which have x and y values. Figure 8b shows a diagram illustrating the relationship between the polar and the cartesian coordinates. Based on these mathematical characteristics, we can easily get the exact edge points of the valid area.



(a) An example of the polar coordinate system

(b) A diagram illustrating the relationship between polar and Cartesian coordinates

Fig. 8 Key concepts of the polar coordinate system

6.2 The DART+ Algorithm for DBRNN query processing

Based on the above key features, DART+ also follows a filter-refinement framework. DART+ utilizes the same index construction and the filtering method as DART. The improvement is applied to the refinement step, and the key features are implemented in the revised refinement step.

Algorithm 6 The refinement step of DART+ for the DBRNN query

```
   Input: A candidate set C
   Output: a set of answers Answer
 1 Answer ← C;
 2 for i ← 0 to size of C do
 3 |   obj ← C[i];
 4 |   obj.polar ←getPolarCoordinate(obj);
 5 |   max.x ←max(polar.x[]);
 6 |   max.y ←max(polar.y[]);
 7 |   min.x ←min(polar.x[]);
 8 |   min.y ←min(polar.y[]);
 9 |   list ←cropping(max.x, max.y, min.x, min.y);
10 |   distance ← getDistance(q, C[i]);
11 |   for j ← 0 to size of list[] do
12 |   |   if distance > getDistance(C[i], list[j]) then
13 |   |   |   angleS ← getAngle(C[i], list[j]);
14 |   |   |   if |(angleS − C[i].d)| ≤ θ/2 then
15 |   |   |   |   Answer.remove(C[i]);
16 |   |   |   |   break;
17 |   |   |   end
18 |   |   end
19 |   end
20 end
21 return Answer
```

6.2.1 Refinement step revisited

Similar to DART, as a result of the filtering step, we have a candidate set that contains all the objects whose valid area contains the query object. In the refinement step, DART+ verifies whether the actual nearest neighbor of each candidate object is the query object more efficiently than DART. Algorithm 6 shows the flow of the refinement step. The same as the refinement step of DART, the method confirms the answer set by checking the nearest neighbor of each candidate. For each candidate object p, if there exists an advertiser object s_i closer than the query object, it is possible that s_i is the nearest neighbor of p, depending

Table 3 The cardinalities of real datasets

Dataset	Cardinality
UX	19,499
NE	123,593

Table 4 The values of parameters

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)

on whether s_i is in the valid area of p. To efficiently check this, we propose the cropping method that significantly reduces the search space by dynamically filtering the objects in S.

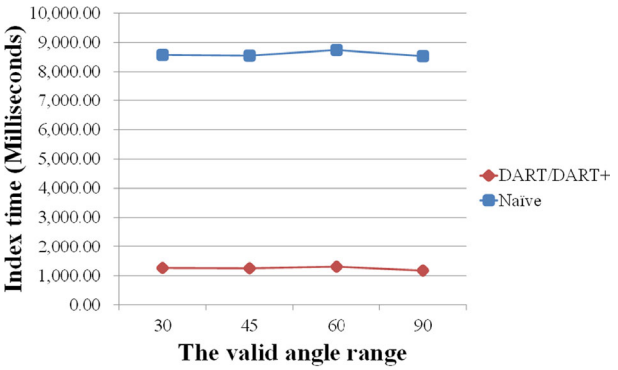
The refinement step iteratively conducts the cropping method for each candidate object. In each iteration, the method tries to get polar coordinates of three points corresponding to the candidate object which are two corner points of the valid area and the center point of the circular arc (Line 5). As we discussed in Section 6.1.2, the two-dimensional coordinates can be easily obtained from the polar coordinates by using the trigonometric functions sine and cosine. With obtained polar coordinates, we find the maximum value and the minimum value for x coordinate and y coordinate to conduct the cropping method (Lines 6-9).

After we get the maximum value and the minimum value for x coordinate and y coordinate from the above procedure, we conduct the cropping method to find a minimum bounding rectangle(MBR) that covers the valid area of the candidate object in P. In this method, we find some grid cells which intersect with the MBR using the assign function which is used in the filtering step. With obtained grid cells, we can retrieve objects that are contained in the grid cells. For the retrieved objects, we verify whether an object is contained in the MBR or not. If an object is within the MBR, then add the object into a list. As a result, we can get a minimized set of objects that can affect to the candidate object during the refinement step. This part is the biggest advantage of DART+, and improves the refinement step of DART.

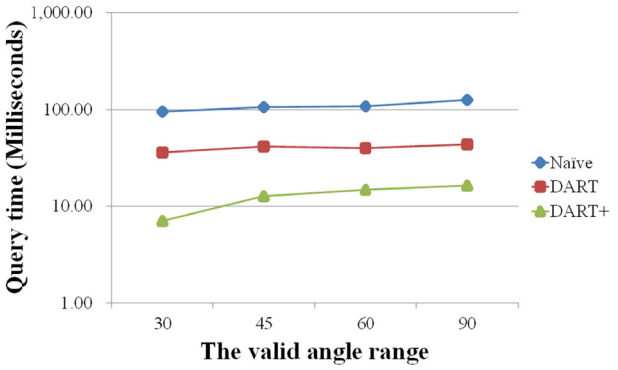
Based on the above cropping method, we conduct the refinement procedure to verify the actual nearest neighbor of the candidate object. For all the candidate object, the method calculates the distance between the candidate object and each remaining advertiser object (Line 12). Moreover, if there is an advertiser object closer than the query object, we check the actual angle degree (Lines 14-17). If the difference is smaller than a half of θ, it means that the directional angle is also facing the object s_i, and the nearest neighbor of the candidate object is not the query object (Lines 14-17). Otherwise, the candidate object can be an answer of the DBRNN query.

6.3 The DART+ Algorithm for DBRkNN query processing

In this section, we extend the algorithm for DBRNN queries to process DBRkNN queries for an arbitrary value k. Similar to DART, although an arbitrary value k is added, the overall process is almost the same. In the refinement step, DART+ should be slightly changed so that k advertiser objects can be checked when finding advertiser objects closer than the query object (Lines 9-10 in Algorithm 6).



(a) Index time for varying valid angle range



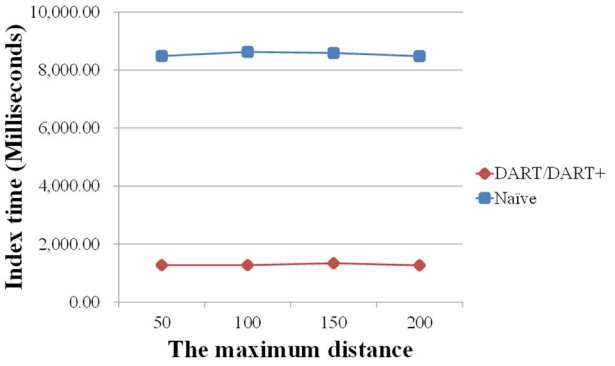
(b) Query time for varying valid angle range

Fig. 9 Experimental results of DBRNN query for varying valid angle range on synthetic datasets

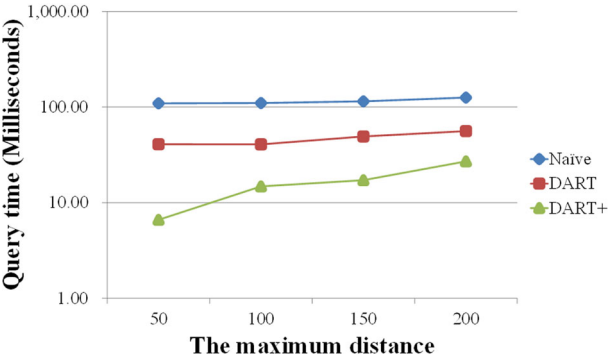
7 Experiments

In this section, we evaluate the performance of our proposed algorithms for the DBRNN query and the DBRkNN query by using synthetic datasets and real 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 10,000 $\times$ 10,000 euclidean space. Moreover, we also use two real datasets, *North East(NE)* dataset and *United States of America and Mexico (UX)* dataset, provided by Chorochronos[1]. The cardinalities of datasets are shown in Table 3. For the real datasets, we consider the objects as advertiser objects, and we add synthetic data for P, because there is no real dataset for the user location with the user's specific heading direction. Moreover, we also normalize the coordinates of the real datasets on the 2-dimensional 1,000,000 $\times$ 1,000,000 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 4.

[1] http://www.chorochronos.org/



(a) Index time for varying maximum distance



(b) Query time for varying maximum distance

Fig. 10 Experimental results of DBRNN query for varying maximum distance on synthetic datasets

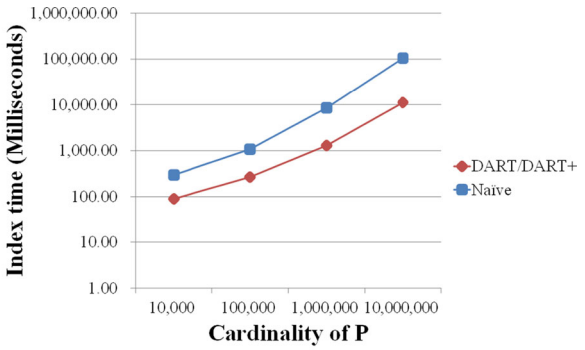
The experiment investigates the index time and query time for varying values of parameters such as the valid angle range, the maximum distance, and the cardinality. All algorithms are implemented in Java, and the experiments are conducted on a PC equipped with Intel Core i7 CPU 3.4GHz and 16GB memory.

7.1 Experimental results of DBRNN query

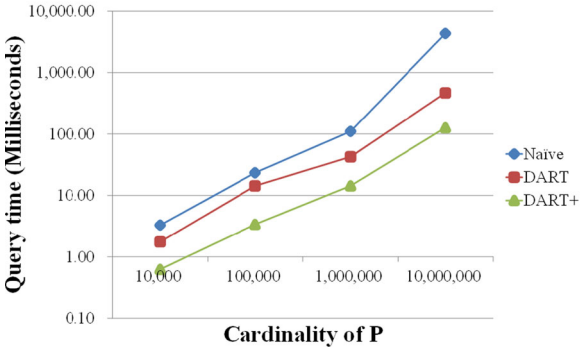
7.1.1 Synthetic datasets

Figures 9, 10, and 11 show the performance of proposed algorithms when processing DBRNN queries with varying values of experimental parameters on the synthetic datasets. Figures 9a, 10a, and 11a represent the index time of each experiment, and Figs. 9b, 10b, and 11b represent the query time. For all results on the index and query processing, DART shows a superior performance compared to the naive method. Moreover, DART+ also outperforms DART.

First, we conduct an experiment with varying the valid angle range varying from 30 degree to 90 degree. According to Fig. 9a, 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



(a) Index time for varying cardinality

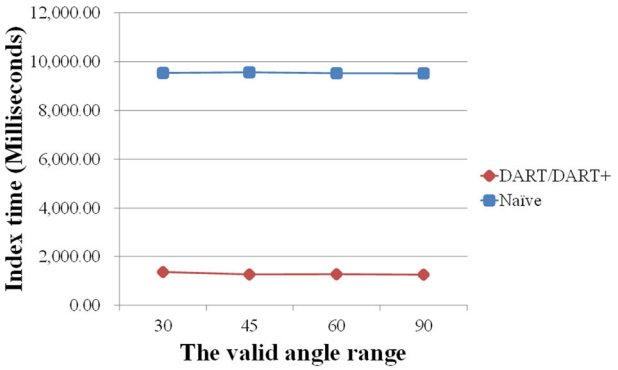


(b) Query time for varying cardinality

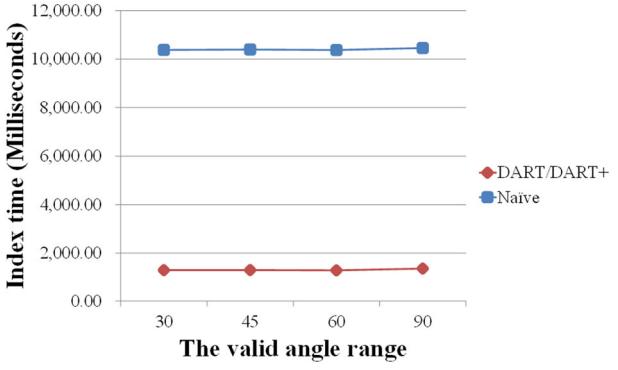
Fig. 11 Experimental results of DBRNN query for varying cardinality on synthetic datasets

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. 9b shows an increasing gap among the query time of the proposed algorithms. 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 and DART+ filter irrelevant objects with its valid angle range, and then conduct 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. In the case of DART+, with the dynamic filtering on the objects in S, DART+ can efficiently remove the objects which do not affect to the candidate objects in P. Therefore, its query time shows an excellent performance because of the further improvement on the refinement step of DART.

According to Fig. 10a, indexing time is almost similar to the result of the valid angle range, and Fig. 10b indicates that the query processing time shows a steady gap



(a) Index time for varying valid angle range on UX



(b) Index time for varying valid angle range on NE

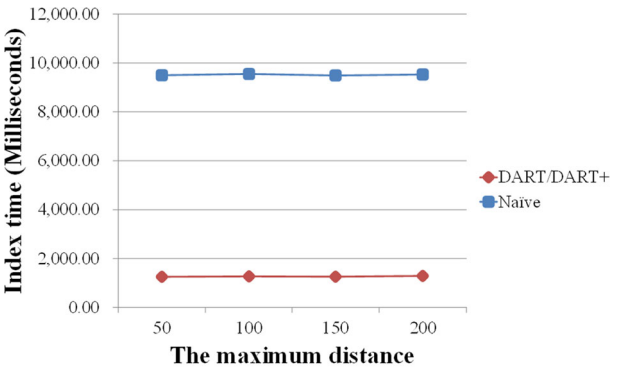
Fig. 12 Experimental results of index time for varying valid angle range on real datasets

among the proposed algorithms. 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 dataset.

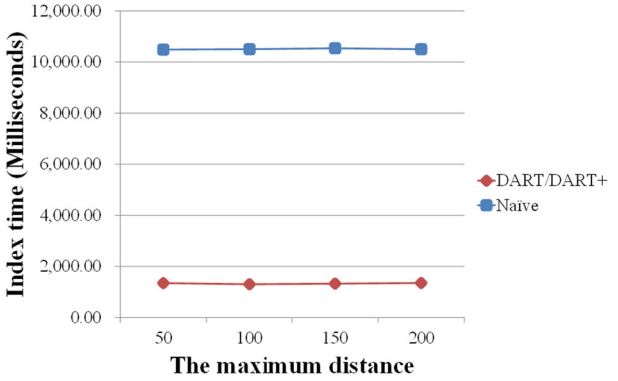
In Fig. 11a and 11b, we conduct experiments with varying the cardinality by using all the datasets. Both index time and query time show similar increasing curves as the cardinality becomes bigger, however, the difference of the performance is upto 10 times among the proposed algorithms. In our observation, DART and DART+ can handle bigger sized datasets more efficiently than the naive method, even though the dataset reaches 10 millions of objects.

7.1.2 Real datasets

We also conduct further experiments with real datasets. Figures 12, 13, and 14 show the performance of proposed algorithms when processing DBRNN queries with varying values of experimental parameters on the real datasets. Figures 12a, b, 13a, b, 14a, and b represent the index time of each experiment. For all results on the index processing, DART



(a) Index time for varying maximum distance on UX



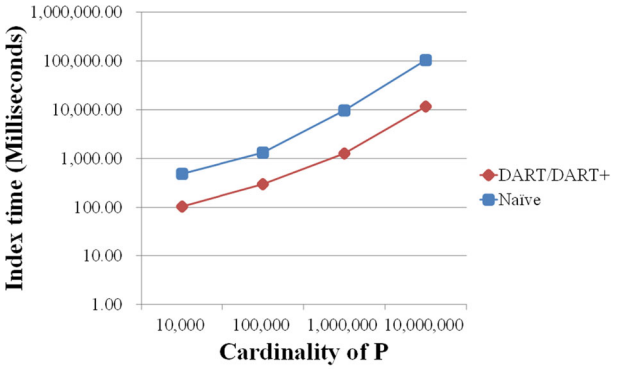
(b) Index time for varying maximum distance on NE

Fig. 13 Experimental results of index time for varying maximum distance on real datasets

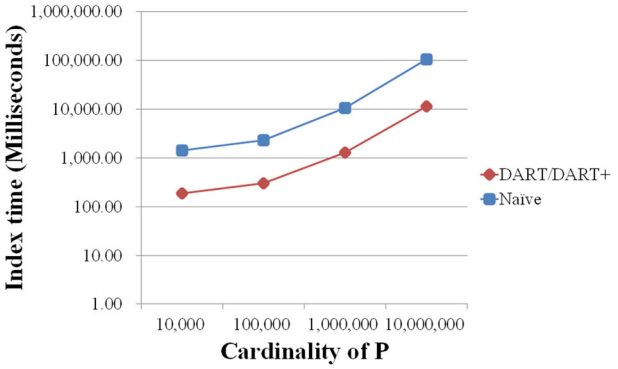
shows a superior performance compared to the naive method, and DART+ also outperforms DART.

For the query processing time, Figs. 15, 16, and 17 show the performance of proposed algorithms when processing DBRNN queries with varying values of experimental parameters on the real datasets. Figures 15a, b, 16a, b, and 17a, b represent the query time of each experiment. For all results on the query processing, DART+ shows an excellent performance.

All results are quite similar to those of the experiments on the synthetic dataset, but there are some differences on the query processing time. In our observation, the main factor of the query processing time is the object density. In the case of the synthetic datasets, the object density is uniformly distributed while the density is not uniformly distributed in the real datasets. In addition, as shown in Table 3, the cardinality of the real dataset is fixed whereas the cardinalities of the synthetic datasets are various depending on the cardinality of P. For these reasons, the result graphs do not show steady lines for the query processing time.



(a) Index time for varying cardinality on UX



(b) Index time for varying cardinality on NE

Fig. 14 Experimental results of index time for varying cardinality on real datasets

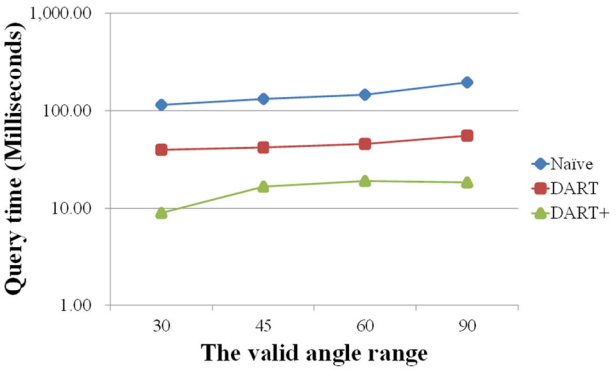
7.2 Experimental results of DBRkNN query

7.2.1 Synthetic datasets

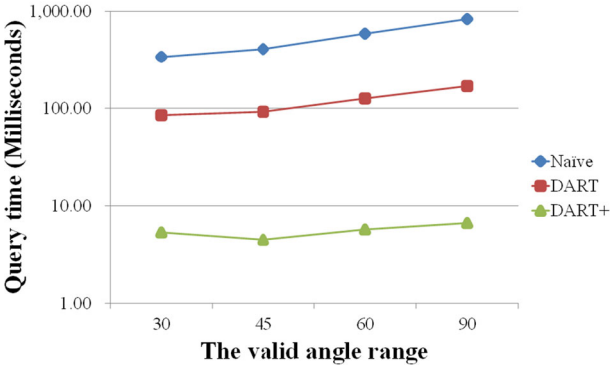
Figure 18 shows the performance of the proposed algorithms for the DBRkNN queries on the synthetic datasets. For the arbitrary k value, we start from $k = 2$ and exponentially increase k until 16. The experimental results show that the query times for the cases are almost uniformly distributed. In our 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 algorithms are also much more efficient for processing the DBRkNN query than the naive method.

7.2.2 Real datasets

Moreover, we also conduct further experiments on the real datasets. Figure 19 shows the performance of the proposed algorithms for the DBRkNN queries on the real datasets. Simi-



(a) Query time for varying valid angle range on UX



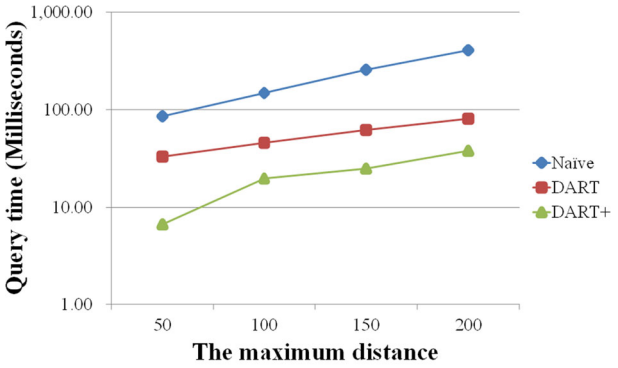
(b) Query time for varying valid angle range on NE

Fig. 15 Experimental results of DBRNN query time for varying valid angle range on real datasets

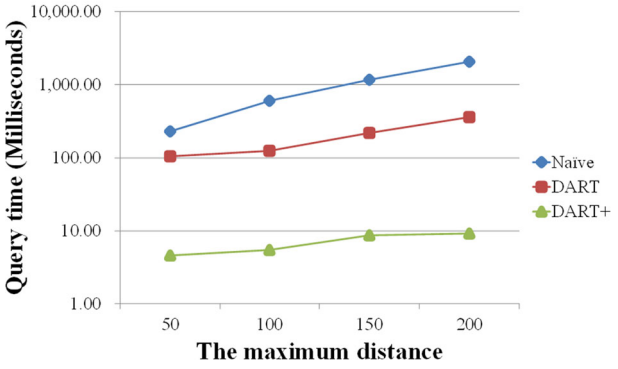
lar to the above experiments, the experimental results show that the query times for the cases are almost uniformly distributed. Moreover, DART+ shows the fastest query time among the proposed algorithms.

7.3 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.



(a) Query time for varying maximum distance on UX



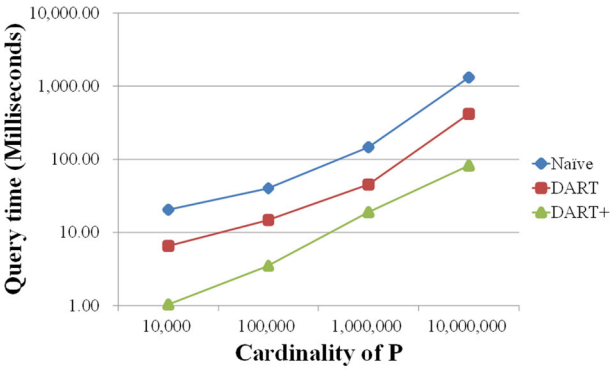
(b) Query time for varying maximum distance on NE

Fig. 16 Experimental results of DBRNN query time for varying maximum distance on real datasets

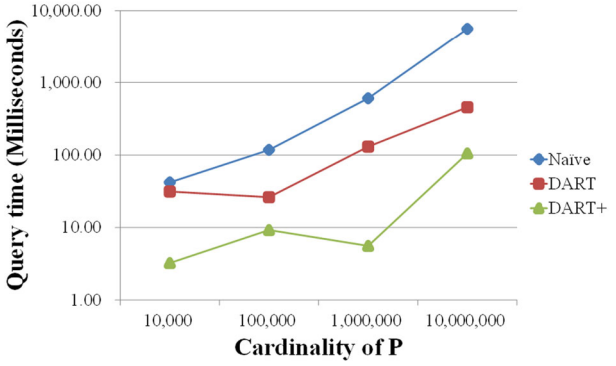
Furthermore, we also have shown that DART+ outperforms DART in both indexing time and query processing time. We conducted several experiments the same as the above, and the results show that DART+ significantly improves the refinement process of DART. However, we found that the object density can affect to the performance of the query processing by observing the experimental results on real datasets. In the case of high density datasets, the performance of DART+ decreases and the gap between DART and DART+ is getting narrow. In our observation, if the objects in S are densely located around the query object and the candidate objects, the cropping method prunes only a small amount of objects. Therefore, the performance of DART+ can be worse than the situation with sparse distribution of the objects. However, as shown in the experiments as the real datasets, the objects appears to be sparse in the real situation.

8 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



(a) Query time for varying cardinality on UX



(b) Query time for varying cardinality on NE

Fig. 17 Experimental results of DBRNN query time for varying cardinality on real datasets

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

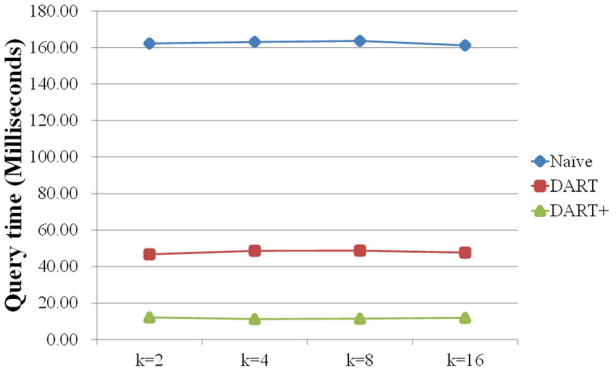
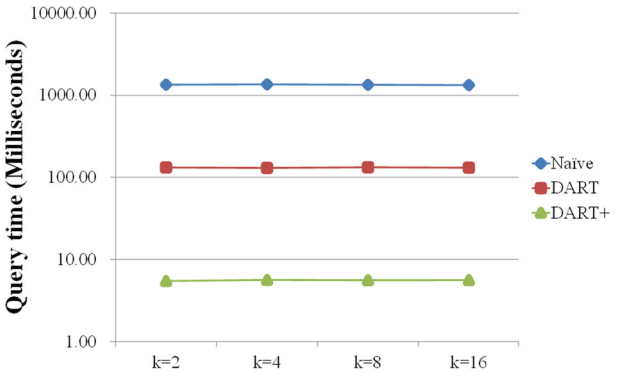
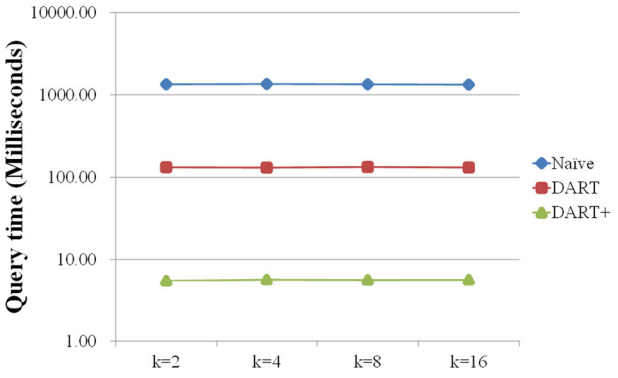


Fig. 18 Experimental results of DBR*k*NN query on synthetic datasets



(a) Query time for varying k value on UX



(b) Query time for varying k value on NE

Fig. 19 Experimental results of DBRkNN query on real datasets

By further extending DART, we propose a more improved version of DART, called DART+, which significantly improves the refinement step of DART by using a dynamic filtering with a cropping method. From extensive experiments, we showed that DART+ also outperforms DART by improving the refinement process.

Acknowledgment We would like to thank the editor and anonymous reviewers. This work was supported in part by Defense Acquisition Program Administration and Agency for Defense Development under the contract UD110006MD, Korea, and in part by the National Research Foundation of Korea(NRF) Grant funded by the Korean Government(MSIP)(No. NRF-2014R1A1A2002499).

References

Achtert, E., Böhm C., Kröger, P., Kunath P., Pryakhin, A., Renz, M. (2006). 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)* (pp. 515–526). New York: ACM.

Benetis, R., Jensen, S., Karciauskas, G., Saltenis, S. (2006). Nearest and reverse nearest neighbor queries for moving objects. *The VLDB Journal*, *15*(3), 229–249.

Cheema, M.A., Zhang, W., Lin, X., Zhang, Y., Li, X. (2012). Continuous reverse k nearest neighbors queries in euclidean space and in spatial networks. *The VLDB Journal*, *21*(1), 69–95.

Dhar, S., & Varshney, U. (2011). Challenges and business models for mobile location-based services and advertising. *Communications of the ACM*, *54*(5), 121–128.

Gao, Y., Zheng, B., Chen, G., Lee, W.C., Lee, K.C.K., Li, Q. (2009). Visible reverse k-nearest neighbor query processing in spatial databases. *IEEE Transactions on Knowledge and Data Engineering*, *21*(9), 1314–1327.

Guttman, A. (1984). R-trees: a dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD international conference on management of data, (SIGMOD '84)* (pp. 47–57). New York: ACM.

Kang, J.M., Mokbel, M.F., Shekhar, S., Xia, T., Zhang, D. (2007). Continuous evaluation of monochromatic and bichromatic reverse nearest neighbors. In *ICDE*.

Korn, F., & Muthukrishnan, S. (2000). Influence sets based on reverse nearest neighbor queries. In *Proceedings of the 2000 ACM SIGMOD international conference on management of data, (SIGMOD '00)* (pp. 201–212). New York: ACM.

Korn, F., Muthukrishnan, S., Srivastava, D. (2002). Reverse nearest neighbor aggregates over data streams. In *Proceedings of the 28th international conference on Very Large Data Bases, VLDB Endowment, (VLDB '02)* (pp. 814–825).

Krumm, J. (2011). Ubiquitous advertising: the killer application for the 21st century. *Pervasive Computing, IEEE*, *10*(1), 66–73.

Lee, K.W., Choi, D.W., Chung, C.W. (2013). Dart: an efficient method for direction-aware bichromatic reverse k nearest neighbor queries. In *SSTD* (pp 295–311).

Li, G., Feng, J., Xu, J. (2012). Desks: direction-aware spatial keyword search. In A. Kementsietsidis & M.A.V. Salles (Eds.), *ICDE* (pp. 474–485). IEEE Computer Society.

Lian, X., & Chen, L. (2008). Monochromatic and bichromatic reverse skyline search over uncertain databases. In *Proceedings of the 2008 ACM SIGMOD international conference on management of data, (SIGMOD '08)* (pp. 213–226). New York: ACM.

Mokbel, M.F., & Levandoski, J.J. (2009). Toward context and preference-aware location-based services. In *Proceedings of the 8th ACM international workshop on data engineering for wireless and mobile access, (MobiDE '09)* (pp. 25–32). New York: ACM.

Nutanong, S., Tanin, E., Zhang, R. (2010). Incremental evaluation of visible nearest neighbor queries. *IEEE Transactions on Knowledge and Data Engineering*, *22*(5), 665–681.

Qiao, S., Tang, C., Jin, H., Long, T., Dai, S., Ku, Y., Chau, M. (2010). Putmode: prediction of uncertain trajectories in moving objects databases. *Applied Intelligence*, *33*, 370–386.

Qin, C., Bao, X., Roy Choudhury, R., Nelakuditi, S. (2011). Tagsense: a smartphone-based approach to automatic image tagging. In *Proceedings of the 9th international conference on mobile systems, applications, and services, (MobiSys '11)* (pp. 1–14). New York: ACM.

Stanoi, I., Agrawal, D., Abbadi, A.E. (2000). Reverse nearest neighbor queries for dynamic databases. In *ACM SIGMOD workshop on research issues in data mining and knowledge discovery* (pp. 44–53). New York: ACM.

Stanoi, I., Riedewald, M., Agrawal, D., Abbadi, A.E. (2001). Discovery of influence sets in frequently updated databases. In *Proceedings of the 27th international conference on Very Large Data Bases, (VLDB '01)* (pp. 99–108). San Francisco: Morgan Kaufmann Publishers Inc.

Taniar, D., Safar, M., Tran, Q.T., Rahayu, W., Park, J.H. (2011). Spatial network rnn queries in gis. *Computer Journal*, *54*(4), 617–627.

Tao, Y., Papadias, D., Lian, X. (2004). Reverse knn search in arbitrary dimensionality. In *Proceedings of the thirtieth international conference on Very Large Data Bases, VLDB Endowment, (VLDB '04)* (Vol. 30, pp. 744–755).

Tao, Y., Yiu, M.L., Mamoulis, N. (2006). Reverse nearest neighbor search in metric spaces. *IEEE Transactions on Knowledge and Data Engineering*, *18*(9), 1239–1252.

Tao, Y., Papadias, D., Lian, X., Xiao, X. (2007). Multidimensional reverse knn search. *The VLDB Journal*, *16*(3), 293–316.

Tran, Q.T., Taniar, D., Safar, M. (2010). Bichromatic reverse nearest-neighbor search in mobile systems. *IEEE Systems Journal*, *4*(2), 230–242.

Vlachou, A., Doulkeridis, C., Kotidis, Y., Norvag, K. (2011). Monochromatic and bichromatic reverse top-k queries. *IEEE Transactions on Knowledge and Data Engineering*, *23*(8), 1215–1229.

Wang, Y., Gao, Y., Chen, L., Chen, G., Li, Q. (2012). All-visible-k-nearest-neighbor queries. In *DEXA* (Vol. 2, pp. 392–407).