Information Sciences 330 (2016) 274–292

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Reverse k-nearest neighbor search in the presence of obstacles[☆]

Yunjun Gao[a,b,*], Qing Liu[a], Xiaoye Miao[a], Jiacheng Yang[c]

[a] College of Computer Science, Zhejiang University, 38 Zheda Road, Hangzhou 310027, China
[b] The Lab of Big Data Intelligent Computing, Zhejiang University, Hangzhou 310027, China
[c] Department of Computer Science, Columbia University, New York, NY 10027, USA

ARTICLE INFO

Article history:
Received 9 October 2014
Revised 9 August 2015
Accepted 4 October 2015
Available online 17 October 2015

Keywords:
Reverse nearest neighbor
Obstructed reverse nearest neighbor
Obstacle
Query processing

ABSTRACT

In this paper, we study a new form of reverse nearest neighbor (RNN) queries, i.e., *obstructed reverse nearest neighbor* (ORNN) search. It considers the impact of obstacles on the *distance* between objects, which is ignored by the existing work on RNN retrieval. Given a data set P, an obstacle set O, and a query point q in a two-dimensional space, an ORNN query finds from P, all the points/objects that have q as their nearest neighbor, according to the *obstructed distance* metric, i.e., the length of the *shortest path* between two points without crossing any obstacle. We formalize ORNN search, develop effective *pruning heuristics* (via introducing a novel concept of *boundary region*), and propose efficient algorithms for ORNN query processing assuming that both P and O are indexed by traditional data-partitioning indexes (e.g., R-trees). In addition, several interesting variations of ORNN queries, namely, *obstructed reverse k-nearest neighbor* (ORkNN) *search*, *ORkNN search with maximum obstructed distance* δ (δ-ORkNN), and *constrained ORkNN* (CORkNN) *search*, have been introduced, and they can be tackled by extending the ORNN query techniques, which demonstrates the *flexibility* of the proposed ORNN query algorithm. Extensive experimental evaluation using both real and synthetic data sets verifies the effectiveness of pruning heuristics and the performance of algorithms, respectively.

1. Introduction

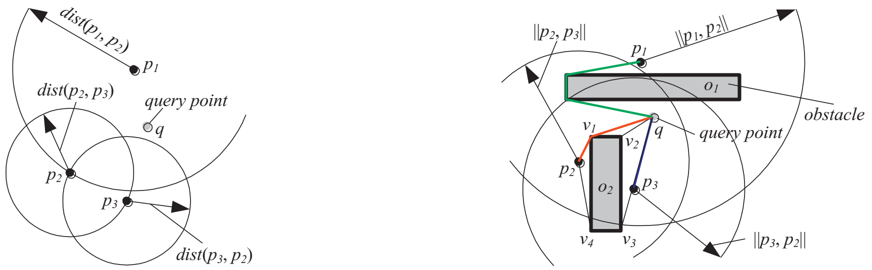
Given a multi-dimensional data set P and a query point q, a *reverse nearest neighbor* (RNN) query retrieves all the points in P that have q as their nearest neighbor (NN). Due to its wide application base such as decision support [15], profile-based marketing [15,26], and resource allocation [15,35], RNN is one of the most popular variants of NN queries [7,12,14,17,20]. Formally, $RNN(q) = \{p \in P \mid q \in NN(p)\}$, in which $RNN(q)$ represents the set of reverse nearest neighbors to q and $NN(p)$ denotes the NN of a point $p \in P$. Consider an example in Fig. 1a, where the data set P consists of three data points (i.e., p_1, p_2, p_3) in a two-dimensional (2D) space. Each point p_i ($1 \leq i \leq 3$) is associated with a vicinity circle/arc $cir(p_i, r)$ centered at p_i and having $r = dist(p_i, NN(p_i))$

☆ This paper is an extended version of the conference paper, titled "On Efficient Obstructed Reverse Nearest Neighbor Query Processing", which has been published in the Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (ACM SIGSPATIAL GIS 2011), November 1–4, 2011, Chicago, IL, USA. Specifically, the paper extends the conference paper by including (i) additional three interesting variants of ORNN queries, i.e., ORkNN search (Section 7.1), δ-ORkNN retrieval (Section 7.2), and CORkNN search (Section 7.2); (ii) enhanced experimental evaluation that incorporates the new classes of queries (Section 8); and (iii) more complete and informative related work (Section 2), more pseudo-codes, more illustrative examples, and more analyzes. More details concerning this paper's extension have also been pointed out explicitly in Section 1 of the paper.

* Corresponding author at: College of Computer Science, Zhejiang University, 38 Zheda Road, Hangzhou 310027, China. Tel.: +86 571 8765 1613; fax: +86 571 8795 1250.

E-mail addresses: gaoyj@zju.edu.cn (Y. Gao), liuq@zju.edu.cn (Q. Liu), miaoxy@zju.edu.cn (X. Miao), jiachengy@cs.columbia.edu (J. Yang).

http://dx.doi.org/10.1016/j.ins.2015.10.022
0020-0255/



(a) RNN search (b) ORNN search

Fig. 1. Example of RNN and ORNN queries

as its radius, i.e., the vicinity circle/arc $cir(p_i, r)$ covers the NN of p_i. Here, $dist()$ refers to a specified distance metric. As shown in Fig. 1a, for the RNN query issued at a point q that uses the Euclidean distance as the distance metric, its result set $RNN(q) = \{p_1\}$, since q is only inside p_1's vicinity arc $cir(p_1, dist(p_1, p_2))$.

RNN search has been well studied, and many efficient algorithms have been proposed to support RNN query and its variants. A short review of some representative algorithms will be presented in Section 2.1. Existing algorithms employ either the *Euclidean distance* in a *Euclidean space* or the *network distance* in a *road network* to measure the proximity between objects. To the best of our knowledge, all those algorithms do not take into account the existence of *obstacles* (e.g., buildings and blindage). However, obstacles are *ubiquitous* in the real world, and their existence may change the *distance* between objects, and hence affect the final query result. Recently, the impact of obstacles on various research problems has attracted much attention from academy [10,11,13,19,23,32,34,39], and many work have been conducted, by taking the influence of obstacles into consideration. For example, the *spatial clustering in the presence of obstacles* (e.g., COD_CLARANS [29], DBRS_O [30], DBCLuC [38], etc.) is a new research direction for the data mining community formed by considering the impact of obstacles on spatial clustering.

In this paper, we study the impact of obstacles on RNN retrieval in a Euclidean space, and form a new type of RNN queries, namely, *obstructed reverse nearest neighbor* (ORNN) search. Given a data set P, an obstacle set O, and a query point q in a 2D space, an ORNN query finds from P, all the points that take q as their NN, according to the *obstructed distance*, i.e., the distance/length of the shortest path that connects two points without crossing any obstacle. An example is depicted in Fig. 1b, where $P = \{p_1, p_2, p_3\}$ and $O = \{o_1, o_2\}$. To simplify the discussion in this paper, we assume that obstacles are in rectangular shapes, although they could be in any other shape as well.

Let $||p_i, q||$ be the obstructed distance from a point p_i to q, and $ONN(p_i)$ be the *obstructed nearest neighbor* (ONN) of p_i that has the smallest obstructed distance to p_i compared with other points. We associate each point $p_i \in P$ with (i) an arc $arc(p_i, ||p_i, ONN(p_i)||)$ centered at p_i and with radius $||p_i, ONN(p_i)||$, and (ii) its obstructed path to q. For instance, the arc $arc(p_3, ||p_3, p_2||)$ centered at p_3 and having $||p_3, p_2||$ as the radius indicates that p_2 is the ONN to p_3, and the straight line from p_3 to q denotes the obstructed path between them without crossing any obstacle. It is observed that $||p_3, q|| < ||p_3, ONN(p_3) = p_2||$ and $||p_2, q|| < ||p_2, ONN(p_2) = p_3||$, and thus, q is the ONN to both p_2 and p_3, i.e., q's ORNN set $ORNN(q) = \{p_2, p_3\}$. Note that, p_1 is the RNN of q in a *Euclidean space* (see Fig. 1a), but it is not the ORNN of q in an *obstructed space* due to the block of obstacle o_1.

We focus on ORNN search because, it is not only a *challenging* problem from the research point of view, but also very *useful* in many applications. As an example, suppose KFC plans to open a new restaurant and wants to distribute coupons to its potential customers for promotion. Assume that there are some buildings and parks (i.e., obstacles) around the new restaurant, and customers who have the new restaurant as their obstructed nearest restaurant are more likely to visit. Consequently, in order to ensure the effectiveness of the promotion, KFC needs to identify the persons that take the new restaurant as their obstructed nearest restaurant, and distribute coupons to them. In addition, due to the ubiquity of obstacles, the ORNN query is obviously important, as a stand-alone tool or a stepping stone, in location-based services, geographic information systems, and complex spatial data analysis/mining involving obstacles.

In addition to the ORNN query, we also study several interesting variations, i.e., (1) *obstructed reverse k-nearest neighbor* (OR*k*NN) *search*, which retrieves all the points in the dataset P that take a given query point q as one of their obstructed k-nearest neighbors (O*k*NN); (2) *ORkNN retrieval with an obstructed distance threshold* δ (δ-OR*k*NN), which finds the OR*k*NN points that has the obstructed distances to q bounded by a pre-defined threshold δ; and (3) *constrained ORkNN* (COR*k*NN) *search*, which returns the OR*k*NN points in a specified restricted area (defined by the spatial region constraints).

In this paper, we present an efficient solution to tackle the ORNN query, which follows a *filter-refinement* framework and does not require any pre-processing. Moreover, we extend ORNN query algorithm to efficiently handle OR*k*NN, δ-OR*k*NN, and COR*k*NN queries, respectively. In brief, the key contributions of the paper are summarized as follows:

- We formalize ORNN search, a *new* addition to the family of spatial queries in the presence of obstacles.
- We introduce a *new* concept of *boundary region* to facilitate the pruning of unqualified data points and node entries.
- We develop efficient algorithms to answer *exact* or *approximate* ORNN retrieval.
- We extend ORNN query techniques to handle several variations of ORNN queries, i.e., OR*k*NN search, δ-OR*k*NN search, and COR*k*NN search.

- We conduct extensive experiments with both real and synthetic data sets to verify the effectiveness of the presented pruning heuristics and the performance of the proposed algorithms.

Note that, this paper extends our preliminary work [9] in several substantial ways. First, we investigate three new ORNN query variants, i.e., OR*k*NN, δ-OR*k*NN, and COR*k*NN queries. Second, we conduct a more comprehensive performance evaluation which incorporates the new classes of queries. Third, we present a more complete review of the related work and more illustrative examples, to make the paper self-contained.

The rest of this paper is organized as follows. Section 2 reviews related work. Section 3 formulates the ORNN query. Section 4 discusses pruning heuristics. Section 5 presents ODC and BRF Algorithms. Section 6 elaborates algorithms for processing ORNN search. Section 7 extends ORNN query solution to tackle several ORNN query variants. Considerable experimental results and our findings are reported in Section 8. Finally, Section 9 concludes the paper with some directions for future work.

2. Related work

In this section, we overview the existing work related to ORNN retrieval, including RNN search, spatial queries with obstacles, and main-memory obstacle path problems.

2.1. RNN queries

Existing algorithms for RNN query and its variants can be classified into three categories. The first category is based on *pre-processing* [15,35]. It pre-computes, for each point p in a given dataset P, the distance from p to its NN p' (i.e., $NN(p)$), and forms a vicinity circle $cir(p, dist(p, p'))$ centered at p and having $dist(p, p')$ as its radius. Then, for a specified query point q, it examines q against all the vicinity circles $cir(p, dist(p, NN(p)))$ with $p \in P$, and those having their vicinity circles enclosing q constitute the final query result, i.e., $RNN(q) = \{p \in P \mid q \in cir(p, dist(p, NN(p)))\}$. To facilitate the examination, all the vicinity circles can be indexed by an RNN-tree [15] or R*d*NN-tree [35]. However, the construction and update costs of the index are *expensive*. Hence, we do not consider the pre-processing based approach.

The second category does not rely on pre-processing but adopts a *filter-refinement* framework [26,25,38]. The filter-refinement framework consists of two steps, i.e., filtering step and refinement step. In the filtering step, the search space is pruned according to the developed pruning heuristics, and a set of candidates is retrieved from the data set. In the refinement step, all the candidates are verified by using NN retrieval criteria and those *false hits* are discarded. The solution to ORNN search also follows the filter-refinement framework, and requires *no* pre-processing.

The third category focuses on a variety of RNN query variants, such as RNN retrieval over moving objects with fixed velocities [3]; RNN queries in metric spaces [1,27], road networks [24], ad-hoc subspaces [36], and large graphs [37], respectively; RNN search on data stream [16], location data [31]; continuous RNN monitoring [6,33], probabilistic RNN search [5,21], ranked RNN query [18], to name just a few.

It is worth pointing out that, all the aforementioned algorithms do not take into account the physical obstacles that are *ubiquitous* in the real world and may affect the *distance* between objects, and thus, they cannot be (directly) applicable to handle ORNN search efficiently.

2.2. Spatial queries with obstacles

The existence of obstacles could affect the *distance* or/and *visibility* between objects. In terms of distance, a suite of algorithms for processing common spatial queries (e.g., range query, NN retrieval) with obstacle constraints have been proposed [36], and a more detailed study of *obstructed* NN search has been conducted in [32]. More recently, *continuous* NN and *moving* k-NN queries in the presence of obstacles are explored in [10,19] as well. In terms of visibility, *visible* NN (VNN) retrieval [23], *visible* RNN (VRNN) search [11], *continuous* VNN retrieval [13], *group* VNN search [34], and *k-maximum visibility* query [22] have been investigated in the literature.

It is worth noting that, both VRNN and ORNN queries consider obstacles. Nonetheless, they are *fundamentally different*. First, they adopt *different distance metrics*. ORNN retrieval employs *obstructed distance* to measure the distance between objects, whereas VRNN search utilizes *Euclidean distance* to indicate the proximity of objects. Second, ORNN retrieval focuses on the impact of obstacles on *distance*, while VRNN search considers the influence of obstacles on *visibility*.

In a word, different from existing works, we aim at handling the RNN query with obstacle constraints. To our knowledge, this paper is the first attempt on this problem.

2.3. Main-memory obstacle path problems

The main-memory based obstacle/shortest path problem in the presence of obstacles has been well-studied in computational geometry [4], and the most common approach is based on the visibility graph *VG*. The vertexes/nodes of *VG* correspond to obstacle vertexes or source/destination point p_s/p_e. Two nodes v_i and v_j are connected iff they are *visible* to each other.

Since the shortest path contains only the edges of *VG* (as proved in [4]), a popular and practical shortest path computation method proceeds in two steps. The first step constructs *VG*; the second step computes the shortest path in *VG* using Dijkstra's algorithm [8]. The time and space complexities of the *VG*-based approach are $O(n^2 \log n)$ and $O(n^2)$, respectively. Here, n is the

Table 1
Symbols and description

Notation	Description
P/O	A set of data points p or obstacles o in a two-dimensional space
T_p/T_o	The R-tree on P or O
LVG	A local visibility graph
$\|\|p, p'\|\|$	The obstructed distance between p and p'
$\|p, q\|$	The provisional obstructed distance from p to q
$SP(p, p')$	The shortest obstacle-free path (shortest path for short)
V_p	A p's boundary vertex set
BR_p/BA_p	The boundary region or angle of p
$P(p, q)$	The shortest path from p to q derived based on LVG
$ONN(q)$	The obstructed nearest neighbor of q
$ORNN(q)$	The result set of an ORNN query issued at q

number of nodes in *VG*. Obviously, the method has a *poor scalability* and cannot guarantee the efficiency when a large number of obstacles are considered.

Consider that when p_s and p_e are close to each other, the majority of the obstacles do not affect their shortest path, and hence, building a *complete VG* is *unnecessary*. Consequently, we try to simplify the computation of obstructed distance, by maintaining a *local* visibility graph *LVG* that only includes those obstacles which may contribute to the shortest paths between specified points.

3. Problem formulation

In this section, we formally define the ORNN query, the focus of this paper. Table 1 summarizes the notations used frequently throughout the paper.

Definition 3.1. (Visibility [11]). Given two points p, p' in a data set P and an obstacle set O, p and p' are *visible* to each other iff there is no any obstacle o in O such that the line segment formed by p and p', denoted as $[p, p']$, crosses o.

Definition 3.2. (Obstacle-free Path [10]). Given two points p, p' in a data set P and an obstacle set O, a path $P(p, p') = \{v_0, v_1, v_2, \ldots, v_n, v_{n+1}\}$ connecting p and p' sequentially passes n nodes (i.e., obstacle vertexes), denoted as v_i $(1 \le i \le n)$, with $v_0 = p$ and $v_{n+1} = p'$. $P(p, p')$ is an obstacle-free path (path for short) iff $\forall\, i \in [0, n]$, v_i and v_{i+1} are visible to each other. Its distance $|P(p, p')| = \Sigma_{i \in [0,\, n]}\, dist(v_i, v_{i+1})$.

Definition 3.3. (Obstructed Distance [29]). Given two points p, p' in a data set P, the obstructed distance between p and p', denoted by $||p, p'||$, is the length of the shortest obstacle-free path (shortest path for short) from p to p', denoted as $SP(p, p')$, i.e., $\forall\, P(p, p')$, $|P(p, p')| \ge |SP(p, p')|$. Here, $||p, p'|| = |SP(p, p')|$.

Fig. 1b shows an example. Since $[p_1, q] \cap o_1 \ne \emptyset$ and $[p_2, q] \cap o_2 \ne \emptyset$, both p_1 and p_2 are *invisible* to q. Also, p_3 is *visible* to q as $[p_3, q] \cap (o_1 \cup o_2) = \emptyset$. In Fig. 1b, there are many obstacle-free paths between points p_2 and q, e.g., $\{q, v_1, p_2\}$, $\{q, v_2, v_1, p_2\}$, $\{q, v_2, v_3, v_4, p_2\}$, etc. The path $\{q, v_1, p_2\}$ is the *shortest* among all the obstacle-free paths from p_2 to q, and thus, its length is the obstructed distance between p_2 and q, i.e., $||p_2, q|| = ||v_1, q|| + ||v_1, p_2||$. In addition, we would like to highlight that, the Euclidean distance between any two points p and p' always forms the *lower bound* for their obstructed distance, i.e., $dist(p, p') \le ||p, p'||$ holds. Based on Definition 3.3, we now formalize ONN and ORNN retrieval in the following Definition 3.4 and Definition 3.5, respectively.

Definition 3.4. (Obstructed Nearest Neighbor). Given a point p outside a data set P and a point p' in P, p' is the *obstructed nearest neighbor* (ONN) of p, denoted by $ONN(p) = p'$, iff $\forall\, p'' \in P$, $||p', p|| \le ||p'', p||$.

Definition 3.5. (Obstructed Reverse Nearest Neighbor Query). Given a data set P, an obstacle set O, and a query point q, an *obstructed reverse nearest neighbor* (ORNN) query retrieves a set $ORNN(q) \subseteq P$ of points that have q as their ONN, i.e., $ORNN(q) = \{p \in P \mid q \in ONN(p)\}$.

Consider Fig. 1b again. As $ONN(p_2) = ONN(p_3) = \{q\}$, $ORNN(q) = \{p_2, p_3\}$. A naive solution to ORNN retrieval is to perform ONN search [39] for every point in a specified data set P, and then return those points $p \in P$ satisfying $q \in ONN(p)$. Nevertheless, as demonstrated by the experimental results to be presented in Section 8, this approach is *very inefficient*, since it has to traverse the data set P and the obstacle set O *multiple times* (i.e., $|P|$ times), resulting in *high* I/O and CPU costs, especially when P is larger. To this end, we propose efficient algorithms for ORNN query processing, assuming that both P and O are indexed by R-trees [2]. In particular, the method proposed in this paper follows a *filter-refinement* framework, requires *no pre-processing*, and enables effective *pruning heuristics* (via a novel concept of *boundary region*) to shrink the search space significantly.

4. Pruning heuristics

Before presenting the pruning heuristics for ORNN retrieval, we introduce some concepts that can be used to the development of effective pruning strategies.

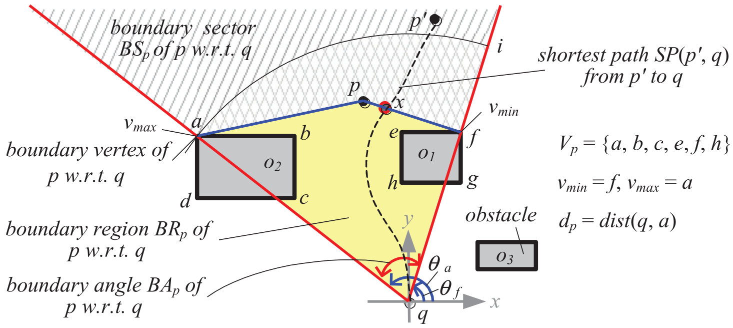


Fig. 2. Illustration of BR_p, BA_p, and BS_p

Definition 4.1. (Point Angle). Given a point p and a query point q, let q be the origin. The amount of rotation (in anti-clockwise direction) about q required to bring the x-axis into correspondence with the line segment $[q, p]$ is defined as p's *point angle* w.r.t. q, denoted by θ_p ($\in [0, 2\pi)$).

Definition 4.2. (Boundary Vertex, Boundary Vertex Set). Given a point p, a vertex v of an obstacle o in an obstacle set O, and a query point q, if v is closer to p than to q according to the obstructed distance, i.e., $||p, v|| < ||q, v||$, v is defined as p's *boundary vertex* w.r.t. q. All p's boundary vertexes w.r.t. q constitute p's *boundary vertex set* V_p w.r.t. q, formally, $V_p = \{v \in o \wedge o \in O \mid ||p, v|| < ||q, v||\}$.

Fig. 2 depicts an example. As $||p, b|| < ||q, b||$, vertex b of obstacle o_2 is p's boundary vertex w.r.t. q; while all the vertexes of obstacle o_3 are closer to q than to p, and thus none of them is p's *boundary vertex* w.r.t. q. Therefore, in Fig. 2, p's *boundary vertex set* $V_p = \{a, b, c, e, f, h\}$ w.r.t. q. In addition, the point angle of vertex f w.r.t. q (i.e., θ_f) is $\angle xqf$, and that of vertex a w.r.t. q (i.e., θ_a) is $\angle xqa$.

Definition 4.3. (Boundary Region, Boundary Angle). Given a point p, a query point q, and p's boundary vertex set V_p w.r.t. q, we define the vertex in V_p having the minimal (maximal) point angle w.r.t. q as v_{min} (v_{max}), i.e., $\exists\, v_{min}, v_{max} \in V_p, \forall\, v \in V_p, \theta_{vmin} \le \theta_v \le \theta_{vmax}$. The *boundary region* of p w.r.t. q, denoted by BR_p, is defined as the polygon formed by the shortest path $SP(p, v_{min})$ from p to v_{min}, the shortest path $SP(p, v_{max})$ from p to v_{max}, the line segment $[q, v_{min}]$, and the line segment $[q, v_{max}]$; and the *boundary angle* of p w.r.t. q, denoted by BA_p, is defined as q's interior angle corresponding to BR_p.

Definition 4.4. (Boundary Sector). Given a point p, a query point q, and p's boundary angle BA_p w.r.t. q, the *boundary sector* of p w.r.t. q, denoted by BS_p, is defined as the circular sector centered at q and with BA_p as its central angle.

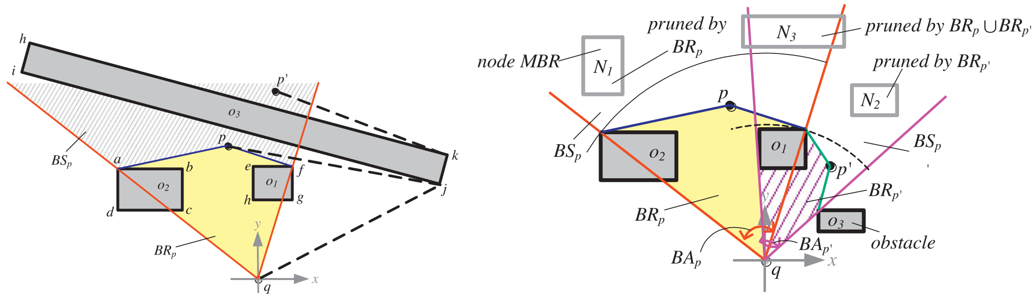
As shown in Fig. 2, since $V_p = \{a, b, c, e, f, h\}$, the vertex v_{min} (v_{max}) with minimal (maximal) point angle w.r.t. q is f (a), i.e., $v_{min} = f$ and $v_{max} = a$. Suppose $SP(p, v_{min}) = \{p, f\}$ and $SP(p, v_{max}) = \{p, a\}$, the polygon formed by $SP(p, v_{min})$, $SP(p, v_{max})$, $[q, v_{min}]$, and $[q, v_{max}]$ is a convex quadrilateral *paqf*, and thus $BR_p = paqf$, $BA_p = \angle fqa$, and p's boundary sector w.r.t. q (i.e., BS_p) is the circular sector centered at q and having $\angle fqa$ as its central angle. It is worth noting that if the polygon formed by $SP(p, v_{min})$, $SP(p, v_{max})$, $[q, v_{min}]$, and $[q, v_{max}]$ is *self-intersecting*, i.e., $SP(p, v_{min})$, $SP(p, v_{max})$, $[q, v_{min}]$, and $[q, v_{max}]$ do not intersect with each other, $BR_p = \emptyset$ and $BA_p = 0$.

The reason for us to introduce the *boundary region* is that it can effectively prune away unqualified data points and node entries, and hence shrink the search space. Take Fig. 2 as an example. Without any auxiliary information, the *entire* search space needs to be scanned. However, once p's boundary region BR_p w.r.t. q (i.e., the polygon *paqf*) is identified, it is guaranteed that any point having the shortest path to q *intersecting* BR_p at *either* $SP(p, v_{min})$ *or* $SP(p, v_{max})$ is closer to p than to q, and thus can be safely discarded, as proved by Lemma 4.1 below

Lemma 4.1. *Given a point p in a data set P, and assume that an ORNN query issued at a query point q, a point $p' \in P$ could not be an ORNN of q if its shortest path to q crosses either $SP(p, v_{min})$ or $SP(p, v_{max})$ at the intersection point x, i.e., $p' \notin ORNN(q)$ if $x \in SP(p', q) \wedge x \in SP(p, v_{min}) \cup SP(p, v_{max})$.*

Proof. Suppose a point p' with its shortest path $SP(p', q)$ to q intersecting either $SP(p, v_{min})$ or $SP(p, v_{max})$ is an ORNN of q. Without loss of generality, we assume $SP(p', q)$ crosses $SP(p, v_{min})$ at point x, i.e., $||p', q|| = |SP(p', q)| = ||p', x|| + ||x, q||$, as illustrated in Fig. 2. Also, we assume that $P(p', p)$ is an *obstacle-free path* from p' to p via x, i.e., $|P(p', p)| = ||p', x|| + ||x, p|| \ge ||p', p||$. On the other hand, as p' is an ORNN of q, $||p', q|| \le ||p', p||$, i.e., $||p', x|| + ||x, q|| = ||p', q|| \le ||p', p|| \le ||p', x|| + ||x, p||$, meaning that $||x, q|| \le ||x, p||$ holds. In other words, $||q, v_{min}|| \le ||x, q|| + ||x, v_{min}|| \le ||x, p|| + ||x, v_{min}|| = ||p, v_{min}||$, i.e., $||q, v_{min}|| \le ||p, v_{min}||$, which contradicts the fact that the vertex v_{min} is a *boundary vertex* of p w.r.t. q. Consequently, the above assumption is invalid, and the proof completes. □

Although Lemma 4.1 can prune certain data points, it incurs *high* CPU overhead since both locating the shortest path $SP(p', q)$ from p' to q and determining the intersection between $SP(p', q)$ and $SP(p, v_{min})$ (or $SP(p, v_{max})$) are *expensive*. Actually, the



(a) False misses (b) Boundary region based pruning

Fig. 3. Illustration of false miss and boundary region based pruning.

CPU cost can be *significantly reduced* if we *slightly relax* the condition of Lemma 4.1. Take Fig. 2 as an example again. Points (e.g., p') locating *inside* the boundary sector BS_p but *outside* the boundary region BR_p are *very likely* to have their shortest paths to q crossing $SP(p, v_{min})$ or $SP(p, v_{max})$, compared with those locating outside BS_p. Consequently, the following Heuristic 1 is developed as the *approximate* implementation of Lemma 4.1.

Heuristic 1. Given a point p in a data set P, and assume that an ORNN query issued at a query point q, it has high probability that a point $p' \in P$ could be pruned by Lemma 4.1 if (i) $p' \in BS_p$ (p's boundary sector w.r.t. q), and (ii) $p' \notin BR_p$ (p's boundary region w.r.t. q).

In general, the pruning heuristic is employed to prune away the points that cannot be in the final result. Nonetheless, if it discards actual answer points, we refer it to as *false misses* (FM). Heuristic 1 prunes all the data points p' that satisfy $p' \in BS_p$ and $p' \notin BR_p$. However, some points filtered out by Heuristic 1 might not have their shortest paths to q intersecting $SP(p, v_{min})$ or $SP(p, v_{max})$. In other words, Heuristic 1 may prune away actual answer points, resulting in *false misses*. Take Fig. 3a as an example. Since the point p' satisfies $p' \in BS_p$ and $p' \notin BR_p$, it can be discarded by Heuristic 1. However, $SP(p', q) = ||pv, k|| + ||k, j|| + ||j, q|| < ||p', k|| + ||k, j|| + ||j, p|| = SP(p', p)$. Hence, the point $p' \in P$ is the ORNN of q. In other words, Heuristic 1 has a *false miss* because it prunes away the real ORNN point p'. In order to quantify the FM, we introduce a metric, i.e., FM ratio, which is the ratio of the number of real answer points pruned by Heuristic 1 to the size of the complete answer set. As reported in the experimental results, the FM ratio is *extremely low*. Since the boundary region BR_p of a point $p \in P$ could be in an *irregular shape*, checking whether a specified point is located outside BR_p is *non-trivial*. To facilitate this examination, Heuristic 2 is proposed.

Heuristic 2. Given a point p in a data set P and a query point q, let d_p be the maximal distance from q to any point along the shortest paths $SP(p, v_{min})$ and $SP(p, v_{max})$, i.e., $\exists\, v' \in SP(p, v_{min}) \cup SP(p, v_{max})$, $\forall\, v \in SP(p, v_{min}) \cup SP(p, v_{max})$, $dist(v, q) \leq dist(v', q) = d_p$. It is confirmed that a point $p' \in P$ satisfies both $p' \in BS_p$ and $p' \notin BR_p$ (i.e., p' is pruned by s) if (i) $\theta_{p'} \in BA_p$ and (ii) $dist(p', q) > d_p$.

Heuristic 2 utilizes an *angular sector* to bound the boundary region in order to further simplify the checking process. As shown in Fig. 2, instead of checking whether a specified point p' is outside the boundary region BR_p of point p w.r.t. q (i.e., quadrilateral *paqf*), Heuristic 2 examines whether the point p' is outside the angular sector *aqi* with d_p as the radius, which only involves the *angle test* and the *distance test*.

Note that Heuristics 1 and 2 can be easily extended to prune *non-leaf node MBRs*. Here, we illustrate the basic idea using Fig. 3b. Specifically, a node MBR (e.g., N_1) that falls into p's boundary sector BS_p w.r.t. q but outside p's boundary region BR_p w.r.t. q could be discarded, because its child entries are very likely to be pruned by Heuristics 1 and 2. Furthermore, in some cases, the pruning of a node MBR requires *multiple* boundary sectors and boundary regions. For instance, the node MBR N_3 in Fig. 3b could be pruned away, since it lies completely in the *union* of BS_p and $BS_{p'}$ but outside the *union* of BR_p and $BR_{p'}$.

The pseudo-code of the *Boundary Region based Pruning Algorithm* (BRP) is presented in Algorithm 1. BRP can perform adaptively, according to the application requirement (i.e., whether the *performance* or the *accuracy* is more important). If approximation is allowed, it employs Heuristics 1 and 2 to prune unqualified data points and node MBRs; otherwise, it uses Lemma 4.1 to conduct exact pruning. Note that, the metric $mindist(N, q)$ in the line 10 of Algorithm 1 denotes the minimal Euclidean distance between a node MBR N and a specified query point q.

5. ODC and BRF Algorithms

In order to enable the boundary region based pruning, there are two issues we have to address, i.e., (i) obstructed distance computation and (ii) boundary region formation. In what follows, we present corresponding solutions.

Algorithm 1
Boundary Region based Pruning Algorithm (BRP).

Input: an entry e (data point p' or node MBR N), a query point q, an identified boundary region set S_{BR} accepting entries in the form $\langle p, BA_p, d_p, SP(p, v_{min}), SP(p, v_{max})\rangle$
Output: TRUE if e can be pruned or FALSE otherwise
if e is a data point p' **then**
  for each entry $\langle p, BA_p, d_p, SP(p, v_{min}), SP(p, v_{max})\rangle \in S_{BR}$ **do**
    if approximation is allowed **then**
      if $\theta_{p'} \in BA_p$ and $dist(p', q) > d_p$ **then** // Heuristics 1, 2
        return TRUE // e can be discarded
    else // exact ORNN search
      if $SP(p', q)$ crosses $SP(p, v_{min})$ or $SP(p, v_{max})$ **then**
        return TRUE // Lemma 4.1
if e is a node MBR N **then**
  get N's boundary angle θ based on Definition 4.3 assuming that the boundary vertex set contains N's four vertexes, and $d = mindist(N, q)$
  if approximation is allowed **then**
    let set S contain all the entries $\langle p, BA_p, d_p, SP(p, v_{min}), SP(p, v_{max})\rangle \in S_{BR}$ with $\theta \cap BA_p \neq \emptyset$
    if $\theta \subseteq \cup_{BAp \in S} BA_p$ and $d > \text{MAX}_{dp \in S}\, d_p$ **then**
      return TRUE // N can be pruned away
return FALSE

5.1. Obstructed distance computation

Existing approaches of obstructed distance computation maintain a *global* visibility graph VG and invoke shortest path algorithms (e.g., Dijkstra's algorithm [8]) to calculate the obstructed distance. However, as mentioned in Section 2.3, these methods are *inefficient*, incurring *high* space complexity and update cost. Therefore, we adopt an *incremental* approach to slowly expand a *local* visibility graph, denoted as LVG, containing the obstacles that may affect the obstructed distance between a given query point q and currently evaluated points. It is worth mentioning that the vertexes of LVG correspond to obstacle vertexes. Two nodes v_i and v_j are connected iff they are *visible* to each other. As demonstrated by Lemma 5.1, as long as we carefully tune a threshold γ and include all the obstacles having their minimal Euclidean distances to q bounded by γ in LVG, it is guaranteed that the shortest path derived based on the current LVG is the *real shortest path*. In addition, we strive to reduce the number of obstructed distance computation by reusing known obstructed distances.

Lemma 5.1. *Given a point p, a query point q, a local visibility graph LVG with radius γ, and assume that all the obstacles o in an obstacle set O having their minimal Euclidean distances (i.e., mindist) to q bounded by γ are contained in LVG, i.e., $\{o \in O \mid mindist(o, q) \leq \gamma\} \subseteq LVG$, and let $P(p, q)$ be the shortest path from p to q derived based on LVG. If $|P(p, q)| \leq \gamma$, $P(p, q)$ must be the real shortest path from p to q, i.e., $P(p, q) = SP(p, q)$ and $|P(p, q)| = \|p, q\|$.*

Proof. If $P(p, q)$ is not the real shortest path from p to q, there must be another one $P_1(p, q) = SP(p, q)$ with $|P_1(p, q)| < |P(p, q)|$. Since $P(p, q)$ is the shortest one among all the paths from p to q such that they only pass the vertexes of obstacles included in LVG, $P_1(p, q)$ must pass at least one vertex v of some obstacle o that is not contained in LVG, i.e., $v \notin LVG$ and $mindist(o, q) > \gamma$. We further partition $P_1(p, q)$ into two paths via v, i.e., $P_{11}(p, v)$ and $P_{12}(v, q)$. As $|P_1(p, q)| = |P_{11}(p, v)| + |P_{12}(v, q)|$, $|P_1(p, q)| > |P_{12}(v, q)| \geq dist(v, q) \geq mindist(o, q) > \gamma$. On the other hand, $|P(p, q)| \leq \gamma$ holds, and hence, $|P_1(p, q)| > |P(p, q)|$ satisfies, which contradicts the assumption above. Thus, the proof completes. □

Algorithm 2 shows the pseudo-code of the *Obstructed Distance Computation Algorithm* (ODC). It divides all the vertexes in LVG into two categories: (i) the set S_{vr} of vertexes that have the *real* obstructed distances to a specified query point q so far, and (ii) the set S_{vv} of vertexes whose obstructed distances to q need to be verified later. Initially, to reduce the number of obstructed distance computation, ODC calculates, for a point p being evaluated, the *provisional* obstructed distance from p to q, denoted by $|p, q|$, based on the current LVG and known obstructed distances (lines 2-3). If $|p, q| \leq \gamma$, it is confirmed that $|p, q|$ is the *actual* obstructed distance, i.e., $|p, q| = \|p, q\|$, according to Lemma 5.1 (line 4). Otherwise (i.e., $|p, q| > \gamma$), the algorithm expands LVG, and employs Dijkstra's algorithm to compute the obstructed distances until $|p, q| \leq \gamma$ holds (lines 5–18). Specifically, ODC extends γ to γ' by using the equation $\gamma' = \gamma + \alpha(|p, q| - \gamma)$, and expands LVG accordingly by calling the GetObs algorithm [13] that can find all the obstacles with their Euclidean distances to q falling inside the range $[\gamma, \gamma']$ and store them in the set S_o (lines 6 and 7). Due to the space limitation, we omit the details of GetObs algorithm. The addition of new obstacles may affect the visibility of the vertexes in S_{vv} (but not the ones in S_{vr}), and thus their obstructed distances to q. In particular, for each vertex $v' \in S_{vv}$, ODC distinguishes three cases: (i) If the adjacent edges of v' in the current LVG intersect any obstacle in S_o, the algorithm deletes v' from S_{vv}, and adds it to the set S_{vc} for the evaluation later (lines 10 and 11). (ii) If the adjacent edges of v' in the current LVG do not cross any obstacle in S_o and $|v', q| \leq \gamma$ holds, the algorithm deletes v' from S_{vv}, and adds it to S_{vr} as $|v', q| = \|v', q\|$ by Lemma 5.1 (line 13). (iii) If the adjacent edges of v' in the current LVG do not intersect any obstacle in S_o and $|v', q| > \gamma$ satisfies, the algorithm deletes v' from S_{vv}, and adds it to S_{vc} (line 14). Thereafter, ODC inserts all the vertexes in S_{vc} to LVG, computes, for p and all the vertexes in S_{vc}, the obstructed distances to q using Dijkstra's algorithm, and moves all the vertexes in S_{vc} to S_{vv} (lines 15–17). Finally, the algorithm adds, for the reuse later, all the vertexes $v''' \in S_{vv}$ to S_{vr} if $|v''', q| \leq \gamma$ holds (lines 19 and 20).

Algorithm 2
Obstructed Distance Computation Algorithm (ODC).

Input: a data point p, a query point q, a local visibility graph *LVG* with radius γ, an obstacle R-tree T_o, a min-heap H_o, the set S_{vr} of vertexes that have the real obstructed distances to q so far, the set S_{vv} of vertexes whose obstructed distances to q need to be verified in the next round
Output: radius γ
/∗ S_{vc}: the set of vertexes whose obstructed distances to q need to be calculated in this round. ∗/
1: initialize $|p, q| = \infty$, $S_o = S_{vc} = \emptyset$ // $|p, q|$: provisional distance to q
2: **for** each vertex $v \in S_{vr}$ and v is visible to p **do**
3: **if** $dist(p, v) + ||v, q|| < |p, q|$ **then** $|p, q| = dist(p, v) + ||v, q||$
4: **if** $|p, q| \le \gamma$ **then** return // $||p, q|| = |p, q|$ by Lemma 5.1
5: **while** $|p, q| > \gamma$ **do**
6: $\gamma' = \gamma + \alpha(|p, q| - \gamma)$ // the value of α is a natural number
7: GetObs($T_o, H_o, q, \gamma', S_o$) // algorithm of [13]
8: $\gamma = \gamma'$, and add the vertexes of every obstacle in S_o to S_{vc}
9: **for** each vertex $v' \in S_{vv}$ **do**
10: **if** the adjacent edges of v' in *LVG* cross any obstacle in S_o **then**
11: $S_{vv} = S_{vv} - \{v'\}$ and $S_{vc} = S_{vc} \cup \{v'\}$
12: **else**
13: **if** $|v', q| \le \gamma$ **then** $S_{vv} = S_{vv} - \{v'\}$ and $S_{vr} = S_{vr} \cup \{v'\}$
14: **else** $S_{vv} = S_{vv} - \{v'\}$ and $S_{vc} = S_{vc} \cup \{v'\}$
15: add all the vertexes in S_{vc} to *LVG*
16: compute the obstructed distances to q using Dijkstra's algorithm for p and all vertexes in S_{vc} based on *LVG*
17: **for** each vertex $v'' \in S_{vc}$ **do** $S_{vc} = S_{vc} - \{v''\}$ and $S_{vv} = S_{vv} \cup \{v''\}$
18: $S_o = \emptyset$ // for the next round
19: **for** each vertex $v''' \in S_{vv}$ **do**
20: **if** $|v''', q| \le \gamma$ **then** $S_{vv} = S_{vv} - \{v'''\}$ and $S_{vr} = S_{vr} \cup \{v'''\}$
21: return γ

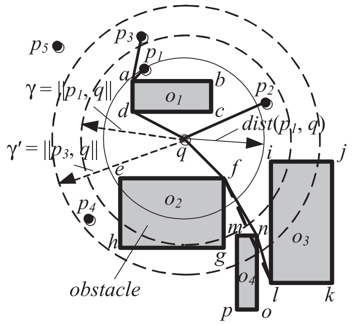


Fig. 4. Example of ODC.

It is worth mentioning that α used in line 6 of Algorithm 2 is a tuning parameter whose value is a natural number. It is introduced to accelerate the expanse of *LVG*. As demonstrated by the experimental results, a *small* value of α reduces the *LVG* size but incurs *high* obstructed distance computation overhead, whereas a *larger* value of α decreases the cost of obstructed distance calculation but increases the size of *LVG*. Nonetheless, how to select the appropriate values of α is one of our future work.

Example 1. We illustrate ODC algorithm with the example depicted in Fig. 4, where the data set $P = \{p_1, p_2, p_3, p_4, p_5\}$ and the obstacle set $O = \{o_1, o_2, o_3, o_4\}$. Suppose $\alpha = 1$ and p_1 is the data point evaluated currently, with $LVG = \{q, p_1\}$, $\gamma = 0$, $S_{vr} = \{q\}$, and $S_{vv} = \emptyset$. First, ODC obtains $|p_1, q| = dist(p_1, q)$ due to $S_{vr} = \{q\}$. As $|p_1, q| > \gamma = 0$, the algorithm performs *while-loop* in lines 5–18 of Algorithm 2, in which it sets γ to $dist(p_1, q)$, calls GetObs to get $S_o = \{o_1, o_2\}$ and $S_{vc} = \{a, b, c, d, e, f, g, h\}$, updates *LVG* to $\{q, p_1, a, b, c, d, e, f, g, h\}$, and utilizes Dijkstra's algorithm, for p_1 and all the vertexes in S_{vc}, to compute all the obstructed distances to q. Note that, in this round of the *while-loop*, ODC skips *for-loop* in lines 9–14 of Algorithm 2 due to $S_{vv} = \emptyset$. Then, the algorithm proceeds in the similar manner until $|p_1, q| \le \gamma$ holds, after which $LVG = \{q, p_1, a, b, c, d, e, f, g, h, i, j, k, l\}$, $\gamma = ||p_1, q|| = dist(p_1, a) + dist(a, d) + dist(d, q)$, $S_{vr} = \{q, a, b, c, d, e, f\}$, and $S_{vv} = \{g, h, i, j, k, l\}$. Finally, vertex i is also added to S_{vr} because $|i, q| = dist(i, q) < ||p_1, q||$. Here, the algorithm terminates, with $LVG = \{q, p_1, a, b, c, d, e, f, g, h, i, j, k, l\}$, $\gamma = ||p_1, q||$, $S_{vr} = \{q, a, b, c, d, e, f, i\}$, and $S_{vv} = \{g, h, j, k, l\}$.

5.2. Boundary region formation

Boundary region formation is based on boundary vertexes. Recall that, for a point p evaluated currently, when identifying p's boundary vertexes w.r.t. a given query point q, we need to find not only the obstructed distance from a vertex v to q but also that from v to p. According to Lemma 5.2, we need to maintain a *new* local visibility graph centered at p, denoted as LVG_p, and tune its radius γ_p so that the *local* shortest path $P(v, p)$ derived based on LVG_p has the *real* shortest path. However, it is not

Algorithm 3
Boundary Region Formation Algorithm (BRF).

Input: a data point p, a query point q, a local visibility graph LVG with radius γ, a boundary region set S_{BR}
Output: a boundary region set S_{BR}
1: initialize $V_p = \emptyset$
2: Dijkstra(LVG, p)
3: **for** each vertex $v \in LVG$ **do**
4:   **if** $|v, q| \le \gamma$ and $|p, v| \le \gamma - dist(v, q)$ and $|p, v| < |v, q|$ **then**
5:     $V_p = V_p \cup \{v\}$ // v is a boundary vertex w.r.t. q
6: **if** $V_p \neq \emptyset$ **then**
7:   find v_{min} and v_{max} from V_p, and obtain BA_p based on Definition 4.3
8:   $d_p = \mathrm{MAX}_{v' \in SP(p,\, vmin) \cup SP(p,\, vmax)}\, dist(v', q)$
9:   $S_{BR} = S_{BR} \cup \langle p, BA_p, d_p, SP(p, v_{min}), SP(p, v_{max})\rangle$
10: return S_{BR}

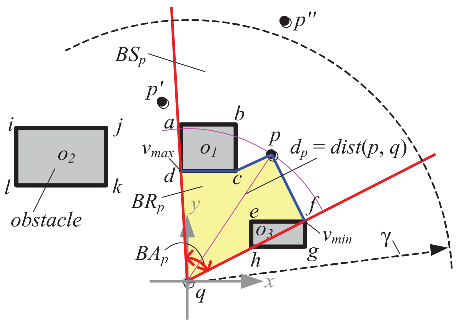


Fig. 5. Example of BRF.

desirable to construct LVG_p since p keeps changing. Moreover, maintaining both LVG_p and LVG (centered at q) is *inefficient* because they contain a large number of common vertexes. Motivated by these findings, in this paper, we only maintain LVG. As proved in Lemma 5.2, the shortest path $P(v, p)$ derived based on LVG with $|P(v, p)| \le \gamma - dist(p, q)$ is the *real shortest path* from v to p.

Lemma 5.2. *Given a point p, a query point q, a local visibility graph LVG (centered at q) with radius γ, and let P(v, p) be the shortest path from v to p derived based on LVG. If $|P(v, p)| \le \gamma - dist(p, q)$, P(v, p) must be the real shortest path from v to p, i.e., $P(v, p) = SP(v, p)$ and $|P(v, p)| = \|v, p\|$.*

Proof. According to Lemma 5.1, a path $P(v, p)$ with $|P(v, p)| \le \gamma - dist(p, q)$ is an *actual shortest path* if *all* the obstacles with their minimal distances (i.e., *mindist*) to p bounded by $(\gamma - dist(p, q))$ are included in current LVG. Assume that there is at least one obstacle o with $mindist(o, p) \le (\gamma - dist(p, q))$ is not contained in the current LVG, i.e., $\exists\, o \in O, mindist(o, p) \le (\gamma - dist(p, q)) \wedge o \notin LVG$. Let v' be a vertex of o such that $dist(v', p) = mindist(o, p)$. If points v', p, and q form a triangle, $mindist(o, q) \le dist(v', q) < dist(v', p) + dist(p, q) = mindist(o, p) + dist(p, q) \le \gamma - dist(p, q) + dist(p, q) = \gamma$. Otherwise, points v', p, and q are located along a line segment, and hence $mindist(o, q) \le dist(v', q) = dist(v', p) + dist(p, q) \le \gamma$. Thus, $mindist(o, q) \le \gamma$ holds. Since the radius of LVG is γ, obstacle o is certainly included in LVG, and the assumption that o is outside LVG is invalid. The proof completes. □

It is worth noting that $P(p, q)$ is updated when LVG expands. According to Lemma 5.2, the minimum $P(p, q)$ is just the real shortest path. A straightforward approach of boundary region formation is to scan the *entire* obstacle set and identify *all* the obstacle vertexes that are closer to the point p (evaluated currently) than to a query point q, in order to constitute p's *complete* boundary vertex set V_p w.r.t. q. Nonetheless, the formation of p's boundary region BR_p w.r.t. q does not necessarily require complete V_p. Take Fig. 2 as an example. The BR_p is formed based on current $V_p = \{a, b, c, e, f, h\}$, although V_p might contain many other boundary vertexes. In addition, complete V_p requires a *global* visibility graph. As mentioned earlier, we only maintain a *local* visibility graph LVG. Consequently, we need to form the boundary region based on LVG.

Algorithm 3 depicts the pseudo-code of the *Boundary Region Formation Algorithm* (BRF). Initially, BRF computes the obstructed distances from the currently evaluated point p to all other vertexes in current LVG, using Dijkstra's algorithm (line 2). Then, for each vertex v in LVG, the algorithm determines whether v is ps boundary vertex w.r.t. q based on Lemma 5.2 and Definition 4.2, and added it to V_p if yes (lines 3–5). In the sequel, the boundary region is formed, which is denoted as a *five-tuple* vector $\langle p, BA_p, d_p, SP(p, v_{min}), SP(p, v_{max})\rangle$ (lines 6–9).

Example 2. We illustrate BRF algorithm with the example shown in Fig. 5, where the data set $P = \{p, p', p''\}$ and the obstacle set $O = \{o_1, o_2, o_3\}$. Assume that p is the data point evaluated currently, with $LVG = \{q, p, a, b, c, d, e, f, g, h, i, j, k, l\}$, $S_{BR} = \emptyset$, and γ. BRF first utilizes Dijkstra's algorithm to calculate the obstructed distances from p to all other vertexes in LVG, and then, obtains $V_p = \{b, c, d, e, f\}$ since they satisfy all the three conditions presented in the line 4 of Algorithm 3. Next, BRF finds $v_{min} = f$ and

Algorithm 4
ORNN Search Algorithm (ORNN).

Input: a data R-tree T_p, an obstacle R-tree T_o, a query point q
Output: the result set S_r of an ORNN query
initialize $S_c = S_r = \emptyset$ and $LVG = \{q\}$
$S_c \leftarrow$ ORNN-Filter (T_p, T_o, q, S_c, LVG) // Algorithm 5
$S_r \leftarrow$ ORNN-Refinement (T_p, T_o, q, S_c, S_r) // Algorithm 6
return S_r

Algorithm 5
Filter for ORNN Algorithm (ORNN-Filter).

Input: a data R-tree T_p, an obstacle R-tree T_o, a query point q, a candidate set S_c, a local visibility graph LVG
Output: a candidate set S_c
/* H_p, H_o: min-heaps; T_p.*root*: the root of T_p; T_o.*root*: the root of T_o */
1: initialize $\gamma = 0$, $S_{BR} = S_{vv} = \emptyset$, $S_{vr} = \{q\}$
2: initialize $H_p = (T_p.root, 0)$, $H_o = (T_o.root, 0)$
3: **while** H_p is not empty **do**
4: de-heap the top entry (e, *mindist*(e, q)) of H_p
5: **if** e is a data point p **then**
6: **if** not BRP (p, q, S_{BR}) **then** // Algorithm 1
7: add p to LVG
8: $\gamma \leftarrow$ ODC ($p, q, LVG, \gamma, T_o, H_o, S_{vr}, S_{vv}$) // Algorithm 2
9: $S_c = S_c \cup \{p\}$
10: $S_{BR} \leftarrow$ BRF ($p, q, LVG, \gamma, S_{BR}$) // Algorithm 3
11: delete p from LVG
12: **else** // e is an intermediate node
13: **for** each child entry $e_i \in e$ **do**
14: **if** not BRP (e_i, q, S_{BR}) **then**
15: insert (e_i, *mindist*(e_i, q)) into H_p
16: **if** approximation is allowed **then**
17: **if** $\cup_{p \in SBR} BA_p = 2\pi$ and $\text{MAX}_{p \in SBR}\, d_p \le mindist(e, q)$ **then**
18: break // terminate algorithm
19: return S_c

$v_{max} = d$, gets p's boundary angle $BA_p = \angle fqd$, derives $d_p = dist(p, q)$, and updates the boundary region set S_{BR} to $\langle p, \angle fqd, dist(p, q), \{p, f\}, \{p, c, d\}\rangle$.

6. ORNN query processing

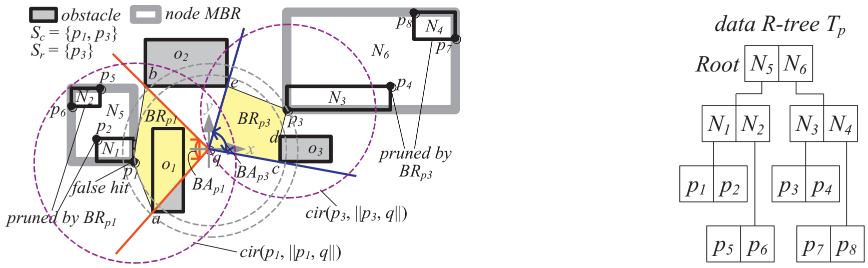
In this section, we explain how to process ORNN search efficiently. Algorithm 4 shows the pseudo-code of the *ORNN Search Algorithm* (ORNN). It follows a *filter-refinement* framework, assuming that the data set P and the obstacle set O are indexed by two *different* R-trees. Specifically, the filtering step prunes unqualified data points and node MBRs using the currently identified boundary regions, and obtains a candidate set S_c which is a *superset* of the final query result set; the subsequent refinement step eliminates the *false hits*.

6.1. The filtering step

Since we rely on the boundary regions for *pruning search space*, *small* boundary regions are preferred. In other words, the boundary regions formed by data points and obstacles that are close to a specified query point q should be identified as early as possible. This is because their corresponding boundary regions are *relatively small*. Consequently, we access data points in *ascending* order of their Euclidean distances (i.e., *mindist*) to q. Two min-heaps H_p and H_o are employed to enable the *best-first* traversal. The pseudo-code of *the Filter for ORNN Algorithm* (ORNN-Filter) is presented in Algorithm 5.

First, ORNN-Filter sets the radius γ of *LVG* to *zero* and initializes the min-heaps H_p and H_o with the root nodes of data R-tree T_p and obstacle R-tree T_o, respectively (lines 1 and 2). Thereafter, it recursively de-heaps the head entry e of H_p for evaluation (lines 3–18). If e is a data point p and cannot be discarded by BRP, ORNN-Filter invokes ODC to compute the obstructed distance between p and q, and then forms p's boundary region BR_p w.r.t. q via BRF. Otherwise, e must be an intermediate (i.e., a non-leaf) node, and the algorithm en-heaps all its child entries for subsequent examination if they cannot be pruned away by BRP.

Note that ORNN-Filter enables an *early termination* if approximation is allowed (lines 16–18). As stated in Heuristic 3 below, when the existing boundary regions stored in the boundary region set S_{BR} span a *full-angle range* (i.e., $\cup_{p \in SBR} BA_p = 2\pi$), the search space for ORNN objects is restricted to a circle *cir*(q, d) centered at q and having $d = \text{MAX}_{p \in SBR}\, d_p$ as its radius. Thus, once the *key* (i.e., *mindist*(e, q)) of the current top entry e of H_p reaches the maximal value of d_p maintained in S_{BR} (i.e., d), it has high probability that the remaining entries (including data points and node MBRs) in H_p cannot become or contain ORNN object(s) based on Heuristics 1, 2, and 3. Note that, Heuristic 3 is based on the boundary region set. If the boundary region set is small, it may not effective.



(a) Data and obstacle placement (b) The R-tree

Fig. 6. Example of ORNN.

Algorithm 6
Refinement for ORNN Algorithm (ORNN-Refinement).

Input: a data R-tree T_p, an obstacle R-tree T_o, a query point q, a candidate set S_c, a result set S_r
Output: a result set S_r
for each candidate $c \in S_c$ **do**
  $ONN_c = \text{ONN}(T_p, T_o, c)$ // algorithm of [39]: find the ONN of c
  if $||ONN_c, c|| \geq ||c, q||$ **then**
    $S_r = S_r \cup \{c\}$ // c is an ORNN of q
return S_r

Heuristic 3. Given an ORNN query issued at a specified query point q, a set S_e of entries (including data points and node MBRs), and a boundary region set S_{BR} with $d = \text{MAX}_{p\in SBR}\, d_p$, it has high probability that the entries in S_e could be pruned by Heuristics 1 and 2 if (i) $\forall\, e \in S_e$, $mindist(e, q) > d$, and (ii) $\cup_{p\in SBR} BA_p = 2\pi$.

Example 3. To facilitate the understanding of ORNN-Filter algorithm, Fig. 6a shows an example with the data set $P = \{p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8\}$, the obstacle set $O = \{o_1, o_2, o_3\}$, the corresponding data R-tree T_p shown in Fig. 6b, and the approximation being allowed. Initially, ORNN-Filter visits the root of T_p and inserts its child entries N_5, N_6 into a min-heap H_p ($= \{N_5, N_6\}$) sorted in ascending order of their *mindist* to a given query point q. Then, the algorithm de-heaps the top entry N_5 of H_p, accesses its child nodes, and en-heaps the entries into $H_p = \{N_1, N_6, N_2\}$. Next, N_1 is visited and it updates H_p to $\{p_1, N_6, p_2, N_2\}$. Since point p_1 is the head entry of H_p and cannot be discarded by BRP (due to $S_{BR} = \emptyset$), ORNN-Filter adds p_1 to a local visibility graph *LVG*, calls ODC to calculate the obstructed distance from p_1 to q, inserts p_1 into S_c ($= \{p_1\}$), utilizes BRF to determine p_1's boundary region BR_{p1} (i.e., polygon qbp_1a) w.r.t. q, updates S_{BR} to $\{\langle p_1, \angle bqa, dist(b, q), \{p_1, b\}, \{p_1, a\}\rangle\}$, and deletes p_1 from *LVG*. The algorithm proceeds in the same manner until the heap H_p becomes empty, with $S_c = \{p_1, p_3\}$. Note that, the de-heaped entries from H_p (including p_2, N_2, p_4, and N_4) during the search are pruned by BR_{p1} or BR_{p3}.

6.2. The refinement step

Once the candidate set S_c is retrieved by ORNN-Filter, the refinement step starts. It validates every candidate via an ONN query [39]. Those candidates that are closer to a given query point q than their obstructed nearest neighbors are returned as the final ORNN objects. Algorithm 6 shows the pseudo-code of the *Refinement for ORNN Algorithm* (ORNN-Refinement).

Example 4. Continue Example 3. Recall that S_c $\{p_1, p_3\}$ after the termination of ORNN-Filter. Now we invoke ORNN-Refinement to verify each candidate in S_c. As shown in Fig. 6, point p_1 is a *false hit* since it is closer to p_2 (i.e., p_1's ONN) than to q, while point p_3 is validated as an actual ORNN of q, and thus it is added to the query result set $S_r = \{p_3\}$.

6.3. Discussion

In the following, we present the time complexity of the ORNN algorithm and prove its correctness.

Let C_{onn} be the cost of an ONN query, and $|P|$, $|O|$, and $|S_c|$ be the cardinality of a data set P, an obstacle set O, and a candidate set S_c, respectively.

Lemma 6.1. *The time complexity of the ORNN algorithm is* $O(|S_c| \times (\log|P| \times |O| \times \log|O| + C_{onn}))$.

Proof. The ORNN algorithm follows the filter-refinement framework, and $|LVG| << |O|$ (as demonstrated by the experimental results to be presented in Section 8). In the filtering step, it takes $O(|S_c| \times \log|P| \times |O| \times \log|O|)$ for obtaining the candidate set S_c; and in the refinement step, it incurs $O(|S_c| \times C_{onn})$ to eliminate all the false hits. Thus, the total time complexity of the ORNN algorithm is $O(|S_c| \times (\log|P| \times |O| \times \log|O| + C_{onn}))$. □

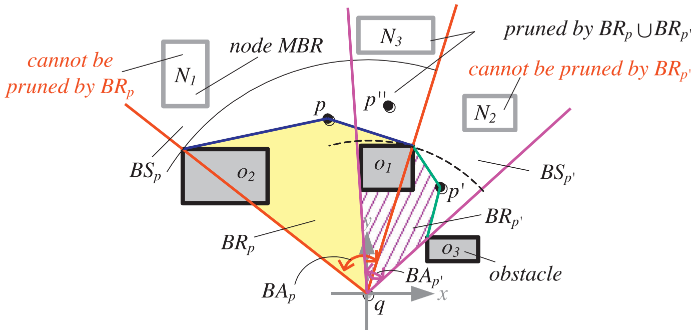


Fig. 7. Example of k-BRP.

It is worth mentioning that, the time complexity of ORNN algorithm, i.e., $O(|S_c| \times (\log|P| \times |O| \times \log|O| + C_{onn}))$, is scalable. This is because $O(|S_c| \times (\log|P| \times |O| \times \log|O| + C_{onn}))$ is log scalable and linear scalable with respect to $|P|$ and $|O|$, respectively. Moreover, the time complexity $O(|S_c| \times (\log|P| \times |O| \times \log|O| + C_{onn}))$ of ORNN algorithm is better than that of the naive ORNN algorithm $O(|P| \times C_{onn})$, which needs to perform an ONN query for every point in P. At the worst case, an ONN query has to traverse the whole $|P|$ and $|O|$ to get the final result, whose time complexity is $C_{onn} = O(|P| \times |O|)$. Hence, $O(|S_c| \times (\log|P| \times |O| \times \log|O| + C_{onn})) = O(|S_c| \times (\log|P| \times |O| \times \log|O| + |P| \times |O|))$ and $O(|P| \times C_{onn}) = O(|P| \times |P| \times |O|)$. As demonstrated by the experimental results, $|S_c| << |P|$. Thus, $O(|S_c| \times (\log|P| \times |O| \times \log|O| + C_{onn}))$ is better than $O(|P| \times C_{onn})$, which can also be confirmed by the experimental results to be presented in Section 8.

Lemma 6.2. *The ORNN algorithm retrieves exactly the ORNNs of a given query point q, i.e., the algorithm has no false negatives and no false positives.*

Proof. First, the ORNN algorithm only prunes away those non-qualifying points or/and node entries in the filtering step, by using the boundary regions identified so far, which guarantees *no false negatives*. Second, every candidate is verified in the refinement step via an ONN query, which ensures *no false positives*. □

Although the ORNN search algorithm presented above assumes that the data set P and the obstacle set O are indexed by two *separate* R-trees, it can be naturally extended to support ORNN search on a *single* R-tree that indexes both data points and obstacles. However, given the focus of this paper and the page limitation we have, we would like to leave that as one of our future work.

7. Extensions

In this section, we extend our techniques to tackle several interesting ORNN query variants, i.e., ORkNN, δ-ORkNN, and CORkNN queries.

7.1. ORkNN search

As defined in Definition 7.2, an ORkNN query aims to find all the points that take a given query point q as one of their OkNN which is defined in Definition 7.1.

Definition 7.1. (Obstructed k-Nearest Neighbor). Given a point p, and a point p' in a data set P, p' is the *obstructed k-nearest neighbor* (OkNN) of p, denoted by $OkNN(p)$, iff there are at most $k - 1$ points r in P satisfying $||r, p|| \le ||p', p||$.

Definition 7.2. (Obstructed Reverse k-Nearest Neighbor Query). Given a data set P, an obstacle set O, and a query point q, an *obstructed reverse k-nearest neighbor* (ORkNN) *query* retrieves a set $ORkNN(q) \subseteq P$ of points that have q as one of their OkNNs, i.e., $ORkNN(q) = \{p \in P \mid q \in OkNN(p)\}$.

Take Fig. 1b as an example, and suppose $k = 2$. Since $O2NN(p_1) = \{p_2, q\}$, $O2NN(p_2) = \{q, p_3\}$, and $O2NN(p_3) = \{q, p_2\}$, $OR2NN(q) = \{p_1, p_2, p_3\}$.

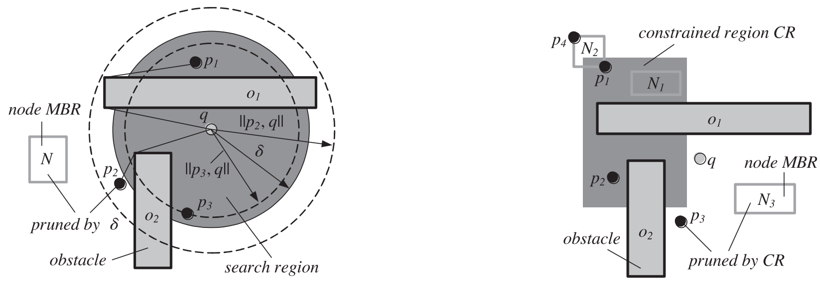
Next, we explain how to extend ORNN query algorithm to answer ORkNN search. First, we discuss how the BRP presented in Algorithm 1 can be extended to the *k-Boundary Region based Pruning Algorithm* (k-BRP). Recall that, for BRP, once a point/MBR falls inside a boundary region w.r.t. q but out of the boundary sector w.r.t. q, it can be discarded shortly. In k-BRP, a point/MBR can be pruned only if it falls into at least k boundary regions. For illustration, we assume $k = 2$ in the following discussion. As depicted in Fig. 7, since point p'' is located within both BR_p and $BR_{p'}$ simultaneously, it can be safely pruned away. For MBR N_3, it can also be discarded as it falls inside the overlap between BR_p and $BR_{p'}$. Nevertheless, N_1 and N_2 cannot be pruned because N_1 only falls into BR_p, and N_2 only falls inside $BR_{p'}$.

Algorithm 7 presents the pseudo-code of k-BRP. Unlike BRP, k-BRP uses a counter to record how many boundary regions a point/MBR falls into (lines 6 and 9). When performing ORkNN retrieval, only when the MBR falls completely inside a pruning

Algorithm 7
k-Boundary Region based Pruning Algorithm (k-BRP).

Input: an entry e (data point p' or node MBR N), a query point q, an identified boundary region set S_{BR} accepting entries in the form $\langle p, BA_p, d_p, SP(p, v_{min}), SP(p, v_{max})\rangle$, a parameter k
Output: TRUE if e can be pruned or FALSE otherwise

```
initialize count = 0 // count: a counter
if e is a data point p′ or a node MBR N then
    for each entry ⟨p, BA_p, d_p, SP(p, v_min), SP(p, v_max)⟩ ∈ S_BR do
        if approximation is allowed then
            if θ_p′ ∈ BA_p and dist(p′, q) > d_p then // Heuristics 1, 2
                count = count + 1
        else // exact ORkNN search
            if SP(p′, q) crosses SP(p, v_min) or SP(p, v_max) then
                count = count + 1 // Lemma 4.1
        if count ≥ k then
            return TRUE // e can be pruned
return FALSE
```



(a) δ-ORNN search (b) CORNN search

Fig. 8. Example of δ-ORNN and CORNN queries.

region, the counter is increased by one. Once the counter reaches k, the corresponding entry can be discarded, and the algorithm returns TRUE.

ORkNN search returns all the points that are closer to a specified query point q than their k-th obstructed nearest neighbor. To solve the ORkNN query, we also follow the *filter-refinement* framework. In particular, we find a set S_c of ORkNN candidates that contains all the *actual* answer points, and then eliminate all the *false* candidates in S_c. The ORNN-Filter algorithm can be easily adapted to support ORkNN retrieval. Recall that, ORNN-Filter enables an early termination if approximation is allowed. To be more specific, when the existing boundary regions stored in S_{BR} span a full-angle range and the *key* of current top entry e of heap H_p reaches the maximal value of d_p maintained in S_{BR} (i.e., $\text{MAX}_{p\in SBR}\ d_p \le mindist(e, q)$), ORNN-Filter stops. However, the early termination condition for ORkNN-Filter is strict. Only when the existing boundary regions stored in S_{BR} span a full-angle range, and meanwhile there are at least k boundary regions for any angle interval, ORkNN-Filter terminates iff the *key* of current top entry e of H_p reaches the maximal value of d_p maintained in S_{BR}. Similarly, the ORNN-Refinement algorithm can also be extended to handle ORkNN search. Nonetheless, we skip the detailed pseudo-codes of ORkNN, due to the similarity between ORNN and ORkNN algorithms, as well as the paper space limitation.

7.2. ORkNN search with constraints

In many real applications, users might enforce some constraints (such as distance and spatial region) on ORkNN queries, and thus, we introduce the ORkNN query with *maximum obstructed distance* δ constraint and *constrained region CR* constraint, respectively, as formally defined in Definition 7.3 and Definition 7.4, respectively.

Definition 7.3. (ORkNN Query with Maximum Obstructed Distance δ). Given a data set P, an obstacle set O, a query point q, and an obstructed distance threshold δ, an *ORkNN query with maximum obstructed distance* δ (δ-ORkNN) returns a set $\delta\text{-}ORkNN(q) \subseteq P$ points, such that $\forall\ p \in \delta\text{-}ORkNN(q)$, $q \in OkNN(p)$ and $||p, q|| \le \delta$, i.e., $\delta\text{-}ORkNN(q) = \{p \in P \mid q \in OkNN(p) \wedge ||p, q|| \le \delta\}$.

An example of δ-ORNN ($k = 1$) search is depicted in Fig. 8a, where p_2 is not the δ-ORNN of q due to $||p_2, q|| > \delta$, while p_3 is a δ-ORNN of q as $||p_3, q|| < \delta$ and $ONN(p_3) = \{q\}$. The δ-ORkNN query has its own applications. Take the KFC application as an example. The manager may only be able to go certain distances to distribute coupons, e.g., not exceeding 2 km. He/she can use the δ-ORkNN query to get the promotion targets.

Definition 7.4. (Constrained Obstructed Reverse k-Nearest Neighbor Query). Given a data set P, an obstacle set O, a query point q, and a constrained region CR, a *constrained obstructed reverse k-nearest neighbor* (CORkNN) query retrieves a set $CORkNN(q) \subseteq P$ points, such that $\forall\ p \in CORkNN(q)$, $q \in OkNN(p)$ and $p \cap CR \ne \emptyset$, i.e., $CORkNN(q)\quad \{p \in P \mid q \in OkNN(p) \wedge p \cap CR \ne \emptyset\}$.

Table 2
Real datasets used in experiments.

Dataset	Cardinality	Description
GR	21,645	2D rivers in Greece
LA	131,461	2D streets in Los Angles

Table 3
Parameter ranges and default values.

Parameter	Range	Default
α	1, 2, 4, 6, 8	4
k	1, 3, 5, 7, 9	5
$\lvert P\rvert/\lvert O\rvert$	0.25, 0.5, 1, 2, 4	1
buffer size (% of the tree size)	0, 5, 10, 15, 20	0
δ (% of the space width)	5, 10, 15, 20, 25	15
CR (% of full space)	10, 20, 30, 40, 50	30

Fig. 8b illustrates an example of CORNN ($k = 1$) search, in which p_3 is not the CORNN of q as it falls outside CR, i.e., $p_3 \cap CR = \emptyset$, while p_2 is a real CORNN point since it is located inside CR with $ONN(p_2) = \{q\}$. Also, COR*k*NN retrieval has its own application base. Consider the KFC application again. The manager wants to distribute coupons within a specified region. In this case, a COR*k*NN query can be employed to decision support.

The proposed algorithms for ORNN search are *flexible*, and can be naturally adjusted to support δ-ORNN and CORNN queries, by integrating *constrained conditions* (i.e., the distance threshold δ and the constrained region CR) during the query processing. In addition, we develop the following heuristics to further boost the search process. First, since the *search region* (SR) of δ-ORNN retrieval is bounded by δ (e.g., the shaded area in Fig. 8a represents the SR of the δ-ORNN query issued at q), the filtering step terminates shortly once the head entry e (a data point or a node) of the heap visited currently has its *mindist* to q (i.e., $mindist(e, q)$) *larger* than δ, because all the remaining entries (e.g., the node MBR N in Fig. 8a) *unvisited* are definitely located outside SR and thus cannot become or contain the actual answer point(s). Moreover, any evaluated data point (e.g., a point p_2 in Fig. 8a) having its obstructed distance to q (i.e., $||p_2, q||$) *exceeding* δ can be directly excluded from any further evaluation. Second, given the fact that the final result of CORNN search must satisfy the specified constrained region CR, any data point or node that does not intersect CR can be pruned away safely, as it cannot become or contain the real answer point(s). In Fig. 8b, for example, a point p_3 and a node MBR N_3 are discarded since they lie outside CR. In addition, the algorithms can also be extended to answer δ-OR*k*NN and COR*k*NN queries, with the extension similar to that for OR*k*NN search (stated earlier), and hence omitted.

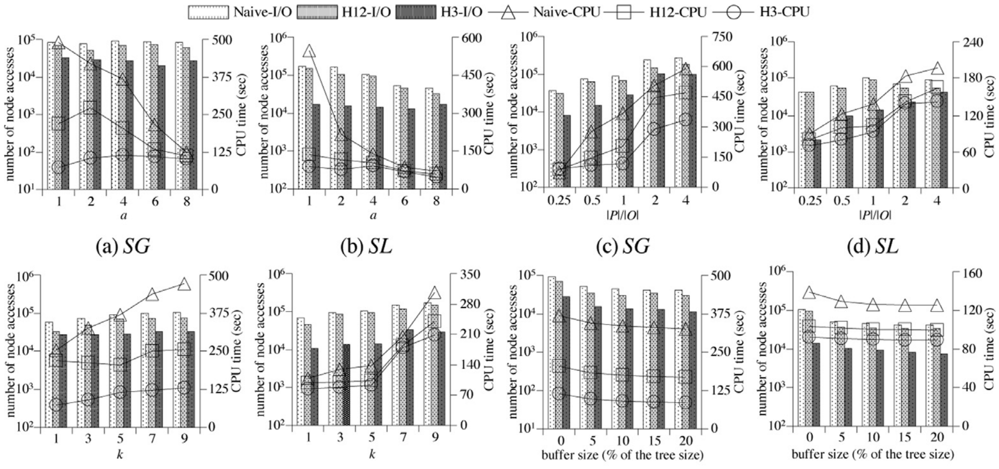
8. Experimental evaluation

In this section, we experimentally evaluate the effectiveness of the developed pruning heuristics and the performance of the proposed algorithms for ORNN search and its variants, using both real and synthetic datasets. All the algorithms were implemented in C++, and all experiments were conducted on an Intel Core 2 Duo 2.93 GHz PC with 3GB RAM.

8.1. Experimental setup

We employ two real datasets, i.e., *GR* and *LA*, which are summarized in Table 2. All the real dataset are available at http://www.rtreeportal.org. We also create several synthetic datasets S_1 and S_2, with their cardinality varying from $0.25 \times |GR|$ to $4 \times |GR|$ and from $0.25 \times |LA|$ to $4 \times |LA|$, respectively. Similar to [39], the distribution of S_1 follows *GR* distribution and that of S_2 follows *LA* distribution. For all datasets, every dimension of the data space is normalized to the range [0; 10,000]. Since OR*k*NN retrieval involves a data set P and an obstacle set O, we deploy two different combinations of the datasets, namely, *SG*, and *SL*, representing $(P, O) = (S_1, GR)$, and (S_2, LA), respectively. It is worth mentioning that the data points in P are allowed to lie on the boundaries of the obstacles but not in their interior. All data and obstacle sets are indexed by R∗-trees [2], with a page size of 4096 bytes.

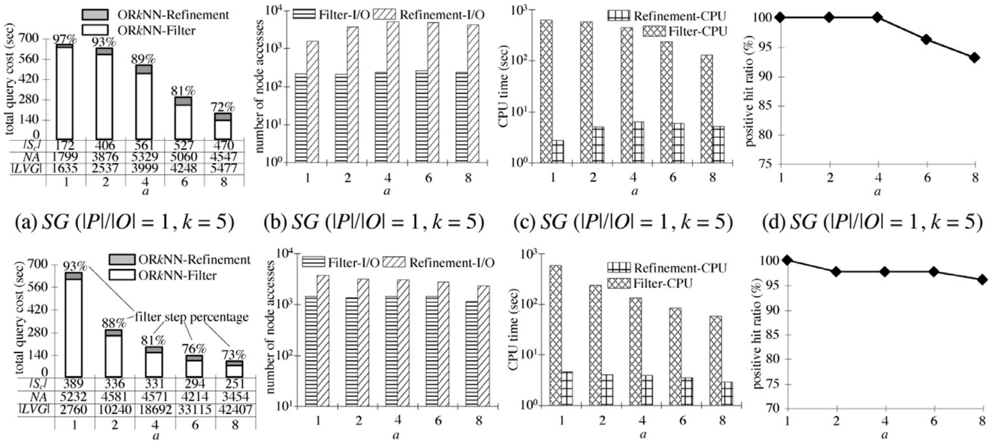
The experiments investigate the performance of the proposed algorithms under a variety of parameters, which are listed in Table 3. It is worth noting that, in each experiment, only one parameter varies, whereas the others are fixed to their default values. The main performance metrics include query cost (i.e., the sum of the I/O time and CPU time, where the I/O time is computed by charging 10 ms for each page access, as with [28]), the cardinality of candidate set S_c (i.e., $|S_c|$), the number of node/page accesses (NA), the CPU time, the local visibility graph size $|LVG|$ (i.e., the number of vertexes contained in LVG), and the *positive hit* (PH) ratio (i.e., the ratio of the number of *actual answer points* to the cardinality of the *complete answer set*). Each reported value in the following diagrams is the average performance of 50 random queries whose locations follow the corresponding obstacle distribution, similar to [39]. Unless specifically stated, the size of LRU buffer is 0 in the experiments, i.e., the I/O cost is determined by the number of node accesses.



(a) SG (b) SL (c) SG (d) SL

(e) SG (f) SL (g) SG (h) SL

Fig. 9. Efficiency of heuristics vs. α, $|P|/|O|$, k, and buffer size, respectively.



(a) SG ($|P|/|O| = 1$, $k = 5$) (b) SG ($|P|/|O| = 1$, $k = 5$) (c) SG ($|P|/|O| = 1$, $k = 5$) (d) SG ($|P|/|O| = 1$, $k = 5$)

(e) SL ($|P|/|O| = 1$, $k = 5$) (f) SL ($|P|/|O| = 1$, $k = 5$) (g) SL ($|P|/|O| = 1$, $k = 5$) (h) SL ($|P|/|O| = 1$, $k = 5$)

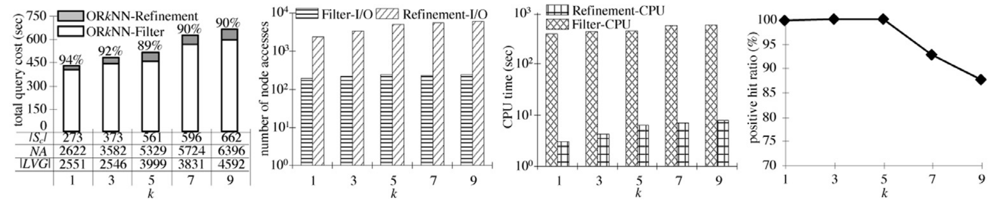
Fig. 10. ORkNN search performance vs. α.

8.2. Effectiveness of pruning heuristics

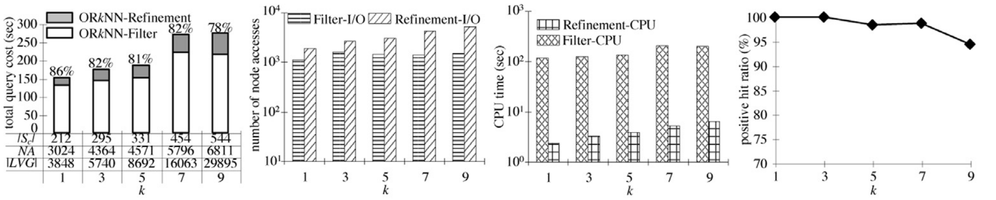
The first set of experiments is to verify the effectiveness of the presented pruning heuristics (i.e., Heuristics 1, 2, and 3). The effectiveness of a heuristic is measured by processing time (i.e., the number of node accesses and CPU time) of the ORkNN algorithms employing different heuristics. Fig. 9 plots the efficiency of different heuristics with respect to α, k, $|P|/|O|$, and buffer size respectively, using dataset combinations *SG* and *SL*. Specifically, Naive denotes the ORkNN algorithm without any heuristic; H12 represents the ORkNN algorithm using Heuristics 1 and 2; and H3 denotes the ORkNN algorithm using Heuristic 3. Notice that the efficiency of Heuristics 1 and 2 is illustrated together with the same curve, since they are applied in the k-BRP algorithm simultaneously. As expected, H12 and H3 outperform Naive significantly. This is because, as pointed out in Section 3, Naive needs to traverse the data R-tree T_p and the obstacle R-tree T_o *multiple times*, incurring *extremely high* I/O overhead and distance computation cost; while H12 and H3 use effective heuristics to shrink the search space significantly. Since the advantage of H12 and H3 over Naive is very *significant*, we only present the experimental results of ORkNN algorithm using Heuristics 1, 2, and 3 in the following presentation, for the clarity of diagrams.

8.3. Results on OR*k*NN queries

The second set of experiments studies the performance of the proposed algorithms for ORkNN queries. First, we investigate the effect of α on the presented ORkNN algorithm, with the experimental results for *SG* and *SL* shown in Fig. 10. Here, the total query cost is broken into two components, corresponding to the filtering step and the refinement step, respectively. The number with percentage on top of each bar indicates the ratio of the cost incurred in the filtering step to that of the overall query cost.

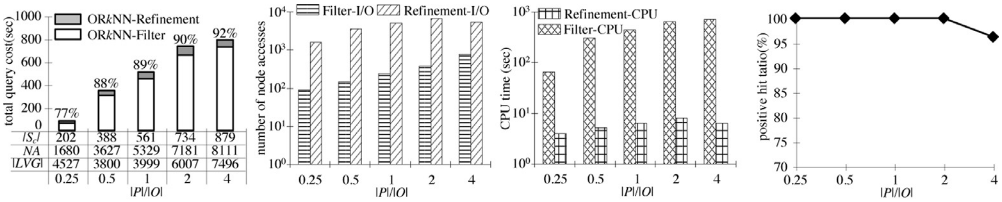


(a) *SG* (α = 4, |P|/|O| = 1) (b) *SG* (α = 4, |P|/|O| = 1) (c) *SG* (α = 4, |P|/|O| = 1) (d) *SG* (α = 4, |P|/|O| = 1)

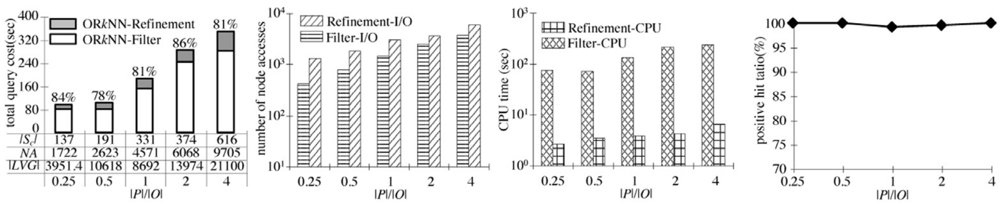


(e) *SL* (α = 4, |P|/|O| = 1) (f) *SL* (α = 4, |P|/|O| = 1) (g) *SL* (α = 4, |P|/|O| = 1) (h) *SL* (α = 4, |P|/|O| = 1)

Fig. 11. OR*k*NN search performance vs. *k*.



(a) *SG* (α = 4, *k* = 5) (b) *SG* (α = 4, *k* = 5) (c) *SG* (α = 4, *k* = 5) (d) *SG* (α = 4, *k* = 5)



(e) *SL* (α = 4, *k* = 5) (f) *SL* (α = 4, *k* = 5) (g) *SL* (α = 4, *k* = 5) (h) *SL* (α = 4, *k* = 5)

Fig. 12. OR*k*NN search performance vs. |*P*|/|*O*|.

Notice that, the cost of OR*k*NN search drops with α. The reason is that, as α becomes larger, the number of obstructed distance computation decreases significantly and the boundary region formation speeds up. Also notice that, although |*LVG*| increases with α, its size is always *much smaller* than the size of data set, as also demonstrated in the subsequent experiments. This is because OR*k*NN algorithm only retrieves *incrementally* the qualified obstacles that may affect the final query result. Fig. 10d and h unveil the PH ratio of the algorithms under different values of α. It is observed that the PH ratio decreases *slightly* as α grows. The reason is that when α is larger (e.g., 8), more obstacles are added to *LVG*, which may speed up the boundary region formation, and thus increase the probability of *false misses*. Nonetheless, the algorithms have *high accuracy*, e.g., the minimal PH ratio is 93.2% as reported in Fig. 10d. The high accuracy can also be observed from all the experiments below.

Second, we explore the impact of *k* on the proposed algorithms, with the results corresponding to *SG* and *SL* depicted in Fig. 11. Note that we fix α at 4, set |*P*|/|*O*| to 1 (i.e., the median value shown in Table 3), and change *k* from 1 to 9. It is observed that, the cost of algorithms increases gradually with the growth of *k*. The reason is that, as *k* grows, the result set enlarges. In other words, more candidates are retrieved in the filtering step, and more candidates are refined in the filtering step, resulting in higher cost.

Third, we study the effect of |*P*|/|*O*| on the proposed algorithms, using *SG* and *SL* dataset combinations. Fig. 12 shows the performance of the algorithms with respect to |*P*|/|*O*|, by fixing α and *k* to 4 and 5 respectively, and changing |*P*|/|*O*| from 0.25 to 4. It is observed that, the cost of algorithms increases gradually with the growth of |*P*|/|*O*|. The reason behind is that, as the density of data set grows, more candidates are retrieved, incurring high refinement overhead.

Finally, we examine the performance of the algorithms in the presence of an LRU buffer, by varying the buffer size from 0% to 20% of the tree size of the dataset *P*. To obtain stable statistics, we perform the first 25 queries to warm up the buffer, and

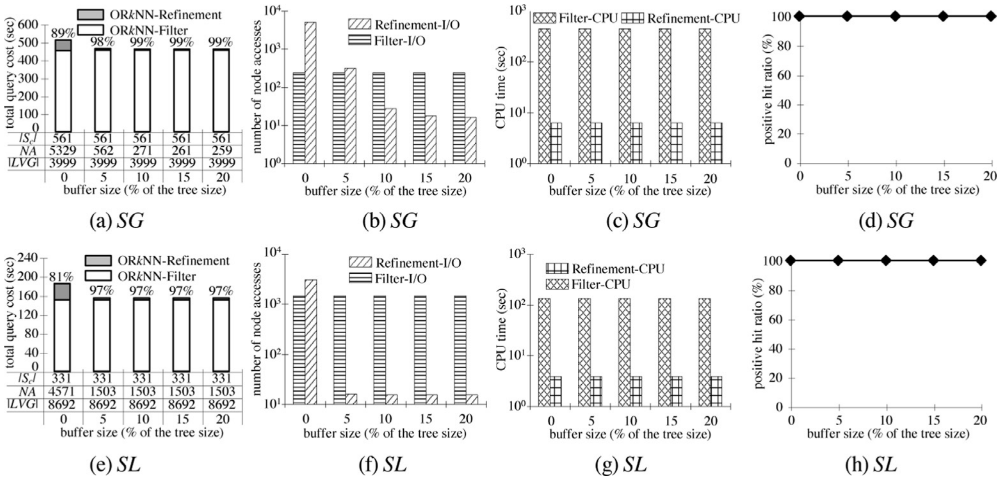


Fig. 13. ORkNN search performance vs. buffer size ($\alpha = 4$, $|P|/|O| = 1$, $k = 5$).

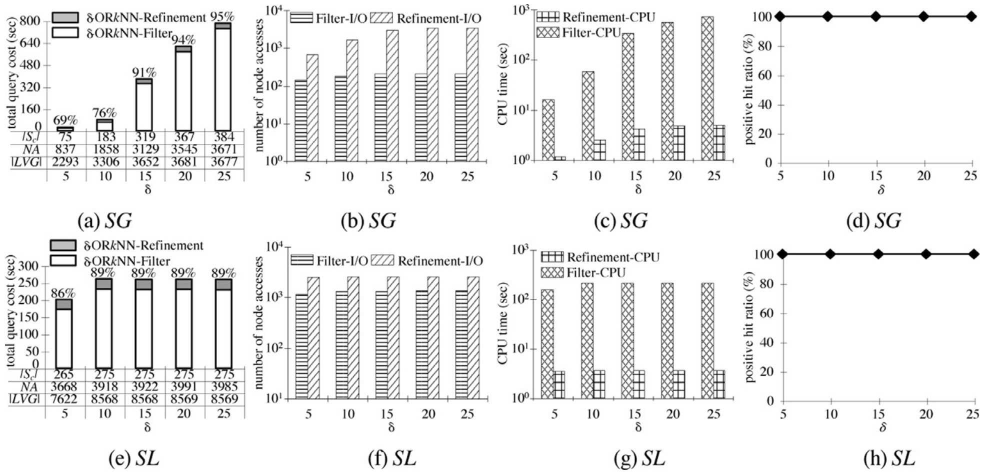


Fig. 14. δ-ORkNN search performance vs. δ ($\alpha = 4$, $|P|/|O| = 1$, $k = 5$).

then report the average cost of the last 25 queries in Fig. 13, with $\alpha = 4$, $k = 5$, and $|P|/|O| = 1$. Note that, when the buffer size enlarges, the cost of ORkNN-Refinement improves although the cost of ORkNN-Filter remains. This is because, the refinement step of ORkNN search requires performing an OkNN query for each candidate, resulting in the traversal of the data R-tree T_p and the obstacle R-tree T_o multiple times. Also notice that the PH ratio remains the same, as shown in Fig. 13d and h, since the buffer stores the nodes visited recently based on LRU, which only affect the performance of algorithms but not the accuracy.

8.4. Results on ORkNN queries with constraints

The last set of experiments evaluates the performance of algorithms for δ-ORkNN and CORkNN queries.

First, we inspect the influence of the maximum obstructed distance δ on the efficiency of δ-ORkNN search algorithm. We change δ values from 5% to 25% of the side length of the search space, and the corresponding results for *SG* and *SL* dataset combinations are depicted in Fig. 14. Clearly, δ has a direct impact on the performance of δ-ORkNN retrieval, since it controls the size of the search region. In particular, the cost of the algorithm increases gradually as δ grows. The reason is that with the growth of δ, the search region enlarges, and thus, more candidate points are retrieved in the filtering step.

We then investigate the impact of the constrained region *CR* size on the performance of CORkNN query processing algorithm. We vary the size of *CR* from 10% to 50% of the whole data space, and present the results in Fig. 15. As expected, the cost of the algorithm increases with the growth of *CR*. This is because, as *CR* grows, more points will fall inside the specified *CR*, and the number of the candidates obtained in the filtering step increases, which leads to more traversals of the obstacle R-tree and more candidate examinations.

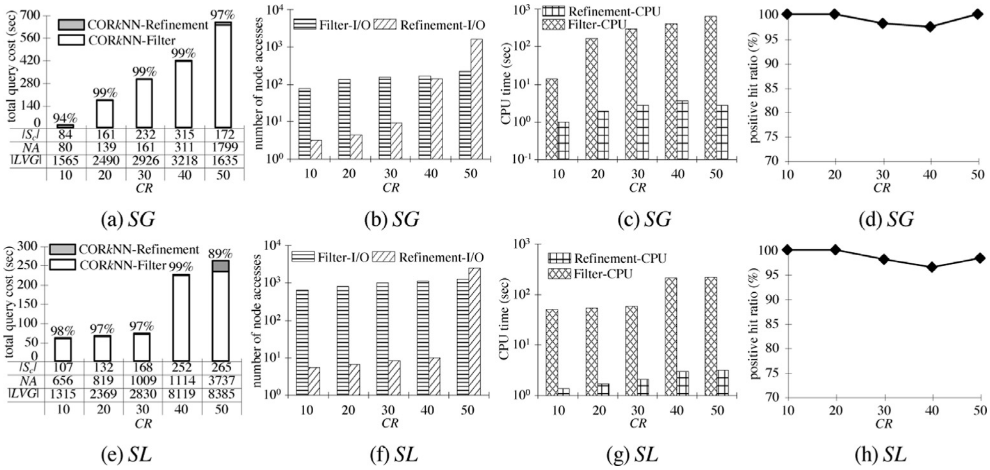


Fig. 15. CORkNN search performance vs. CR ($\alpha = 4$, $|P|/|O| = 1$, $k = 5$).

9. Conclusions

This paper, for the first time, identifies and solves a new type of RNN queries, namely, *obstructed reverse nearest neighbor* (ORNN) search, which considers the impact of obstacles on the *distances* between objects. The ORNN query is not only interesting from a research point of view, but also useful in many decision support applications involving spatial data and physical obstacles. We carry out a systematic study of ORNN retrieval. We carefully formalize the problem, develop effective pruning heuristics by introducing a novel concept of boundary region, propose efficient algorithms for ORNN query processing, extend ORNN query solution to tackle several variations of ORNN queries, i.e., ORkNN, δ-ORkNN, and CORkNN queries, and conduct extensive experiments with both real and synthetic datasets to demonstrate the effectiveness of our presented pruning heuristics and the performance of our proposed algorithms. In the future, we intend to explore the ORkNN search w.r.t. a *line segment* that contains *continuous* query points instead of a *single* query point.

Acknowledgments

We would like to thank Jun Zhang for providing us the source codes proposed in [39]. This work was supported in part by the 973 Program no. 2015CB352502, NSFC Grants no. 61522208, 61379033 and 61472348, and the Fundamental Research Funds for the Central Universities under Grant no. 2015XZZX005-07.

References

[1] E. Achtert, C. Bohm, P. Kroger, P. Kunath, A. Pryakhin, M. Renz, Efficient reverse k-nearest neighbor search in arbitrary metric spaces, in: Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD), 2006, pp. 515–526.

[2] N. Beckmann, H.-P. Kriegel, R. Schneider, B. Seeger, The R∗-tree: an efficient and robust access method for points and rectangles, in: Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD), 1990, pp. 322–331.

[3] R. Benetis, C.S. Jensen, G. Karciauskas, S. Saltenis, Nearest and reverse nearest neighbor queries for moving objects, VLDB J. 15 (3) (2006) 229–249.

[4] M.D. Berg, M.V. Kreveld, M. Overmars, O. Schwarzkopf, Computational Geometry: Algorithms and Applications, Springer-Verlag, second ed., 2000.

[5] M.A. Cheema, X. Lin, W. Wang, W. Zhang, J. Pei, Probabilistic reverse nearest neighbor queries on uncertain data, IEEE Trans. Knowl. Data Eng. 22 (4) (2010) 550–564.

[6] M.A. Cheema, X. Lin, Y. Zhang, W. Wang, W. Zhang, Lazy updates: an efficient technique to continuously monitoring reverse kNN, in: Proceedings of the International Conference on Very Large Data Base (VLDB), 2009, pp. 1138–1149.

[7] J.K. Chen, Y.H. Chin, A concurrency control algorithm for nearest neighbor query, Inf. Sci. 114 (1–4) (1999) 187–204.

[8] E. Dijkstra, A note on two problems in connexion with graphs, Numer. Math. 1 (1959) 269–271.

[9] Y. Gao, J. Yang, G. Chen, B. Zheng, C. Chen, On efficient obstructed reverse nearest neighbor query processing, in: Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (GIS), 2011, pp. 191–200.

[10] Y. Gao, B. Zheng, G. Chen, C. Chen, Q. Li, Continuous nearest-neighbor search in the presence of obstacles, ACM Trans. Database Syst. 36 (2) (2011) 9.

[11] Y. Gao, B. Zheng, G. Chen, W.-C. Lee, K.C.K. Lee, Q. Li, Visible reverse k-nearest neighbor query processing in spatial databases, IEEE Trans. Knowl. Data Eng. 21 (9) (2009) 1314–1327.

[12] Y. Gao, B. Zheng, G. Chen, Q. Li, C. Chen, G. Chen, Efficient mutual nearest neighbor query processing for moving object trajectories, Inf. Sci. 180 (11) (2010) 2176–2195.

[13] Y. Gao, B. Zheng, G. Chen, Q. Li, X. Guo, Continuous visible nearest neighbor query processing in spatial databases, VLDB J. 20 (3) (2011) 371–396.

[14] H.R. Jung, Y.D. Chung, L. Liu, Processing generalized k-nearest neighbor queries on a wireless broadcast stream, Inf. Sci. 188 (2012) 64–79.

[15] F. Korn, S. Muthukrishnan, Influence sets based on reverse nearest neighbor queries, in: Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD), 2000, pp. 201–212.

[16] F. Korn, S. Muthukrishnan, D. Srivastava, Reverse nearest neighbor aggregates over data streams, in: Proceedings of the International Conference on Very Large Data Base (VLDB), 2002, pp. 814–825.

[17] E.-Y. Lee, H.-J. Cho, T.-S. Chung, K.-Y. Ryu, Moving range k nearest neighbor queries with quality guarantee over uncertain moving objects, Inf. Sci. 325 (2015) 324–341.

[18] K.C.K. Lee, B. Zheng, W.C. Lee, Ranked reverse nearest neighbor search, IEEE Trans. Knowl. Data Eng. 20 (7) (2008) 894–910.

[19] C. Li, Y. Gu, F. Li, M. Chen, Moving *k*-nearest neighbor query over obstructed regions, in: Proceedings of the International Asia-Pacific Web Conference (APWEB), 2010, pp. 29–35.
[20] G. Li, L. Li, J. Li, Y. Li, Network voronoi diagram on uncertain objects for nearest neighbor queries, Inf. Sci. 301 (2015) 241–261.
[21] X. Lian, L. Chen, Efficient processing of probabilistic reverse nearest neighbor queries over uncertain data, VLDB J. 18 (3) (2009) 787–808.
[22] S. Masud, F.M. Choudhury, M.E. Ali, S. Nutanong, Maximum visibility queries in spatial databases, in: Proceedings of the International Conference on Data Engineering (ICDE), 2013, pp. 637–648.
[23] S. Nutanong, E. Tanin, R. Zhang, Incremental evaluation of visible nearest neighbor queries, IEEE Trans. Knowl. Data Eng. 22 (5) (2010) 665–681.
[24] M. Safar, D. Ebrahimi, D. Taniar, Voronoi-based reverse nearest neighbor query processing on spatial networks, Multimed. Syst. 15 (5) (2009) 295–308.
[25] A. Singh, H. Ferhatosmanoglu, A. Tosun, High dimensional reverse nearest neighbor queries, in: Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM), 2003, pp. 91–98.
[26] I. Stanoi, D. Agrawal, A.E. Abbadi, Reverse nearest neighbor queries for dynamic databases, in: Proceedings of the ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD), 2000, pp. 44–53.
[27] Y. Tao, M.L. Yiu, N. Mamoulis, Reverse nearest neighbor search in metric spaces, IEEE Trans. Knowl. Data Eng. 18 (9) (2006) 1239–1252.
[28] Y. Tao, D. Papadias, X. Lian, Reverse *k*NN search in arbitrary dimensionality, in: Proceedings of the International Conference on Very Large Data Base (VLDB), 2004, pp. 744–755.
[29] A.K.H. Tung, J. Hou, J. Han, Spatial clustering in the presence of obstacles, in: Proceedings of the International Conference on Data Engineering (ICDE), 2001, pp. 359–367.
[30] X. Wang, H.J. Hamilton, Clustering spatial data in the presence of obstacles, Int. J. Artif. Intell. Tools 14 (1–2) (2005) 177–198.
[31] W. Wu, F. Yang, C.Y. Chan, K.-L. Tan, Finch: Evaluating reverse *k*-nearest-neighbor queries on location data, in: Proceedings of the International Conference on Very Large Data Base (VLDB), 2008, pp. 1056–1067.
[32] C. Xia, D. Hsu, A.K.H. Tung, A fast filter for obstructed nearest neighbor queries, in: Proceedings of the British National Conference on Databases (BNCOD), 2004, pp. 203–215.
[33] T. Xia, D. Zhang, Continuous reverse nearest neighbor monitoring, in: Proceedings of the International Conference on Data Engineering (ICDE), 2006, pp. 203–215.
[34] H. Xu, Z. Li, Y. Lu, K. Deng, X. Zhou, Group visible nearest neighbor queries in spatial databases, in: Proceedings of the International Conference on Web Age Information Management (WAIM), 2010, pp. 333–344.
[35] C. Yang, K.-I. Lin, An index structure for efficient reverse nearest neighbor queries, in: Proceedings of the International Conference on Data Engineering (ICDE), 2001, pp. 485–492.
[36] M.L. Yiu, N. Mamoulis, Reverse nearest neighbors search in ad-hoc subspaces, in: Proceedings of the International Conference on Data Engineering (ICDE), 2006, p. 76.
[37] M.L. Yiu, D. Papadias, N. Mamoulis, Y. Tao, Reverse nearest neighbors in large graphs, IEEE Trans. Knowl. Data Eng. 18 (4) (2006) 540–553.
[38] O.R. Zaiane, C.-H. Lee, Clustering spatial data in the presence of obstacles: a density-based approach, in: Proceedings of the International Database Engineering and Applications Symposium (IDEAS), 2002, pp. 214–223.
[39] J. Zhang, D. Papadias, K. Mouratidis, M. Zhu, Spatial queries in the presence of obstacles, in: Proceedings of the International Conference on Extending Database Technology (EDBT), 2004, pp. 366–384.