Reverse $k$-nearest neighbor search in the presence of obstacles✩

Yunjun Gao, Qing Liu, Xiaoye Miao, Jiacheng Yang

*College of Computer Science, Zhejiang University, 38 Zheda Road, Hangzhou 310027, China
†The Lab of Big Data Intelligent Computing, Zhejiang University, Hangzhou 310027, China
‡Department of Computer Science, Columbia University, New York, NY 10027, USA

A R T I C L E   I N F O

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

A B S T R A C T

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 (OR$k$NN) search, OR$k$NN search with maximum obstructed distance $\delta$ ($\delta$-OR$k$NN), and constrained OR$k$NN (COR$k$NN) 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.

© 2015 Elsevier Inc. All rights reserved.

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, $\text{RNN}(q) = \{ p \in P | q \in \text{NN}(p) \}$, in which $\text{RNN}(q)$ represents the set of reverse nearest neighbors to $q$ and $\text{NN}(p)$ 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 $\text{cir}(p_i, r)$ centered at $p_i$ and having $r = \text{dist}(p_i, \text{NN}(p_i))$.

✩ 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., OR$k$NN search (Section 7.1), $\delta$-OR$k$NN retrieval (Section 7.2), and COR$k$NN 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/© 2015 Elsevier Inc. All rights reserved.
as its radius, i.e., the vicinity circle/arc $\text{cir}(p_i, r)$ covers the NN of $p_i$. Here, $\text{dist}()$ 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 $\text{RNN}(q) = \{p_1\}$, since $q$ is only inside $p_1$’s vicinity arc $\text{cir}(p_1, \text{dist}(p_1, p_2))$.

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 academia [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], DBCLU[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 $\text{ONN}(p_i)$ 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 $\text{arc}(p_i, ||p_i, \text{ONN}(p_i)||)$ centered at $p_i$ and with radius $||p_i, \text{ONN}(p_i)||$, and (ii) its obstructed path to $q$. For instance, the arc $\text{arc}(p_1, ||p_1, p_2||)$ centered at $p_1$ and having $||p_1, p_2||$ as the radius indicates that $p_2$ is the ONN to $p_1$, and the straight line from $p_1$ to $q$ denotes the obstructed path between them without crossing any obstacle. It is observed that $||p_2, q|| < ||p_2, \text{ONN}(p_2)|| = ||p_2, p_3||$, and thus, $q$ is the ONN to both $p_2$ and $p_3$, i.e., $q$’s ORNN set $\text{ORNN}(q) = \{p_2, p_3\}$. 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 (OK$k$NN); (2) OR$k$NN retrieval with an obstructed distance threshold $\delta$ ($\delta$-OR$k$NN), which finds the OR$k$NN points that has the obstructed distances to $q$ bounded by a pre-defined threshold $\delta$; and (3) constrained OR$k$NN (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, $\delta$-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, $\delta$-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., ORNN, 5-ORNN, and CORNN 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 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 ∈ P, and those having their vicinity circles enclosing q constitute the final query result, i.e., RNN(q) = {p ∈ P | q ∈ cir(p, dist(p, NN(p)))}. To facilitate the examination, all the vicinity circles can be indexed by an RNN-tree [15] or RdNN-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 ps/pd. Two nodes vi and vj 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
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 \( v_i \) \((1 \leq i \leq n)\), 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 \( |P(p, p')| = \sum_{i=0}^{n} \text{dist}(v_i, v_{i+1}) \).

**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., \( \forall P(p, p'), |P(p, p')| \geq |SP(p, p')| \). Here, \(||p, p'|| = |SP(p, p')| \).

Fig. 1b shows an example. Since \([p_1, q] \cap o_1 \neq \emptyset \) and \([p_2, q] \cap o_2 \neq \emptyset \), 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 the all 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., \( \text{dist}(p, p') \leq ||p, p'|| \) 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 \( \text{ONN}(p) = p' \), iff \( \forall p'' \in P, ||p'', p'|| \leq ||p', p'|| \).

**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 \( \text{ORNN}(q) \subseteq P \) of points that have \( q \) as their ONN, i.e., \( \text{ORNN}(q) = \{ p \in P \mid q \in \text{ONN}(p) \} \).

Consider Fig. 1b again. As \( \text{ONN}(p_2) = \text{ONN}(p_3) = \{q\} \), \( \text{ORNN}(q) = \{p_2, p_3\} \). 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 \( q \in \text{ONN}(p) \). 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.
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 $\theta_p (\in [0, 2\pi])$.

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 vertices w.r.t. $q$ constitute $p$’s boundary vertex set $V_p$ w.r.t. $q$. Formally, $V_p = \{v \in o \land o \in O \mid ||p, v|| < ||q, v||\}$.

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 vertices of obstacle $o_2$ 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., $\theta_f$) is $\angle qaf$, and that of vertex $a$ w.r.t. $q$ (i.e., $\theta_a$) is $\angle qa$. 

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 $v_{\min}$ (i.e., $v_{\min} = \min_{\forall v \in V_p} \theta_v \land v_{\min} \in V_p$, $\forall v \in V_p, \theta_{v_{\min}} \leq \theta_v \leq \theta_{v_{\max}}$). 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 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 $v_{\min}$ (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 $BR_p = paqf$, $BA_p = \angle qfa$, and $p$’s boundary sector w.r.t. $q$ (i.e., $BS_p$) is the circular sector centered at $q$ and having $\angle qfa$ 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 $[q, v_{\max}]$ do not intersect with each other, $BR_p = \emptyset$ and $BS_p = 0$.

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., $p' \not\in ORNN(q)$ if $x \in SP(p, v_{\min}) \cup SP(p, v_{\max})$.

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., $||p', q|| = ||p', x|| + ||x, q||$, 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., $||P(p', p)|| = ||p', x|| + ||x, p|| \geq ||p', p'||$. On the other hand, as $p'$ is an ORNN of $q$, $||p', q|| \leq ||p, q||$, i.e., $||p', x|| + ||x, q|| = ||p', p'|| \leq ||p, x|| + ||x, q||$, meaning that $||x, q|| \leq ||x, p||$ holds. In other words, $||q, v_{\min}|| \leq ||x, q|| + ||x, v_{\min}|| \leq ||x, q|| + ||x, p||$, $||q, v_{\min}|| = ||p, v_{\min}||$, i.e., $||q, v_{\min}|| \leq ||p, v_{\min}||$, 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
whether a specified point is located outside a specified region. Note that, the metric Heuristic 2.

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) $p' \in BSp$ ($p'$’s boundary sector w.r.t. $q$), and (ii) $p' \notin BRp$ ($p$’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 to it as false misses (FM). Heuristic 1 prunes all the data points $p'$ that satisfy $p' \in BSp$ and $p' \notin BRp$. 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 $p' \in BSp$ and $p' \notin BRp$, 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 $BRp$ of a point $p \in P$ could be in an irregular shape, checking whether a specified point is located outside $BRp$ 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., $\exists \ p' \in SP(p, v_{min}) \cup SP(p, v_{max}), \forall q \in SP(p, v_{min}) \cup SP(p, v_{max}), \text{dist}(v, q) \leq \text{dist}(v', q) = d_p$. It is confirmed that a point $p' \in P$ satisfies both $p' \in BSp$ and $p' \notin BRp$ (i.e., $p'$ is pruned by $s$) if (i) $\theta_p \in BA_p$ and (ii) $\text{dist}(p', q) > d_p$.

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 $BSp$ w.r.t. $q$ but outside $p'$’s boundary region $BRp$ 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 $N_2$ in Fig. 3b could be pruned away, since it lies completely in the union of $BS_p$ and $BS_q$, but outside the union of $BR_p$ and $BR_q$.

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 $\text{mindist}(N, q)$ 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 $(p, BA, d_p, SP(p, v_{min}), SP(p, v_{max}))$

Output: TRUE if $e$ can be pruned or FALSE otherwise

1: if $e$ is a data point $p'$ then
2: for each entry $(p, BA, d_p, SP(p, v_{min}), SP(p, v_{max})) \in S_{BR}$ do
3: if approximation is allowed then
4: if $\theta_{p'} \in BA$ and $d(p', q) > d_p$ then // Heuristics 1, 2
5: return TRUE // $e$ can be discarded
6: else // exact ORNN search
7: if $SP(p', q)$ crosses $SP(p, v_{min})$ or $SP(p, v_{max})$ then
8: return TRUE // Lemma 4.1
9: end if
10: get $N$'s boundary angle $\theta$ based on Definition 4.3 assuming that the boundary vertex set contains $N$'s four vertexes, and $d = \text{mindist}(N, q)$
11: if approximation is allowed then
12: let set $S$ contain all the entries $(p, BA, d_p, SP(p, v_{min}), SP(p, v_{max})) \in S_{BR}$ with $\theta \cap BA \neq \emptyset$
13: if $\theta \subseteq \text{BA}_p$ and $d > \text{MAX}_{d_p} \geq d_p$ then
14: return TRUE // $N$ can be pruned away
15: end if
16: 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 if they are visible to each other. As demonstrated by Lemma 5.1, as long as we carefully tune a threshold $\gamma$ and include all the obstacles having their minimal Euclidean distances to $q$ bounded by $\gamma$ 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 $\gamma$, and assume that all the obstacles $o$ in an obstacle set $O$ having their minimal Euclidean distances (i.e., mindist) to $q$ are contained in $LVG$, i.e., $\{o \in O \mid \text{mindist}(o, q) \leq \gamma\} \subseteq LVG$, 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 $\text{mindist}(o, q) > \gamma$. We further partition $P_1(p, q)$ into two paths via $v$, i.e., $P_1(v, p)$ and $P_2(v, q)$. As $|P_1(p, q)| = |P_1(v, p)| + |P_2(v, q)|$, $|P_1(p, q)| > |P_2(v, q)| \geq \text{dist}(v, q) \geq \text{mindist}(o, q) > \gamma$. 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 $S_{vv}$ 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 $y'$ by using the equation $y' = y + \alpha(|p, q| - \gamma)$, and expands $LVG$ accordingly by calling the GetObjs 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 $S_{vv}$ (lines 6 and 7). Due to the space limitation, we omit the details of GetObjs algorithm. The addition of new obstacles may affect the visibility of the vertexes in $S_{vv}$ (but not the ones in $S_{vv'}$), 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 $S_{vv}$, the algorithm deletes $v'$ from $S_{vv}$, and adds it to the set $S_{vv'}$ 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 $S_{vv}$ and $|v', q| \leq \gamma$ holds, the algorithm deletes $v'$ from $S_{vv}$, and adds it to $S_{vv'}$ 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 $S_{vv}$ and $|v', q| > \gamma$ satisfies, the algorithm deletes $v'$ from $S_{vv}$, and adds it to $S_{vv'}$ (line 14). Thereafter, ODC inserts all the vertexes in $S_{vv'}$ to $LVG$, computes, for $p$ and all the vertexes in $S_{vv'}$, the obstructed distances to $q$ using Dijkstra's algorithm, and moves all the vertexes in $S_{vv'}$ to $S_{vv}$ (lines 15–17). Finally, the algorithm adds, for the reuse later, all the vertexes $v''\in S_{vv'}$ to $S_{vv}$ if $|v''|, q \leq \gamma$ holds (lines 19 and 20).
Algorithm 2
Obstructed Distance Computation Algorithm (ODC).

\begin{algorithm}
\begin{algorithmic}[1]
\Input a data point \( p \), a query point \( q \), a local visibility graph \( LVG \) with radius \( \gamma \), an obstacle \( R \)-tree \( \mathcal{R} \), a min-heap \( H_{\gamma} \), the set \( S_{\gamma} \) of vertexes that have the real obstructed distances to \( q \) so far, the set \( S_{\gamma_0} \) of vertexes whose obstructed distances to \( q \) need to be verified in the next round.
\EndInput
\Output radius \( \gamma \)
\Indp \hrulefill \* \Svv \hrulefill \EndIndent
\State initialize \([p, q] = \infty, S_{\gamma} = S_{\gamma_0} = \emptyset // [p, q]:\) provisional distance to \( q \)
\For {each vertex \( v \in S_{\gamma} \) and \( v \) is visible to \( p \)}
\If {dist\((p, v) + ||v, q|| < ||p, q||\)} \qquad \text{\( [p, q] = \text{dist}(p, v) + ||v, q|| \)}
\EndIf
\EndFor
\If {\([p, q] < \gamma \)} \qquad \text{return} \ // \text{\([p, q] = [p, q] \) by Lemma 5.1}
\EndIf
\While {\([p, q] > \gamma \)}
\State \( \gamma' = \gamma + \alpha[|p, q| - \gamma] \) // the value of \( \alpha \) is a natural number
\EndWhile
\State GetObs\((T_{\gamma}, H_{\gamma}, q', S_{\gamma_0}) // \text{algorithm of [13]}\)
\For {each vertex \( v' \in S_{\gamma_0} \)}
\If {the adjacent edges of \( v' \) in \( LVG \) cross any obstacle in \( S_{\gamma_0} \)}
\State \( S_{\gamma_0} = S_{\gamma_0} - \{v'\} \) and \( S_{\gamma_0} = S_{\gamma_0} \cup \{v'\} \)
\EndIf
\Else
\State \( S_{\gamma_0} = S_{\gamma_0} - \{v'\} \) and \( S_{\gamma_0} = S_{\gamma_0} \cup \{v'\} \)
\EndElse
\EndFor
\State add all the vertexes in \( S_{\gamma_0} \) to \( LVG \)
\EndWhile
\State compute the obstructed distances to \( q \) using Dijkstra’s algorithm for \( p \) and all vertexes in \( S_{\gamma_0} \) based on \( LVG \)
\For {each vertex \( v'' \in S_{\gamma_0} \)}
\If {\([p, q] \leq \gamma \)}
\State \( S_{\gamma_0} = S_{\gamma_0} - \{v''\} \) and \( S_{\gamma_0} = S_{\gamma_0} \cup \{v''\} \)
\EndIf
\EndFor
\State for each vertex \( v'' \in S_{\gamma_0} \) do
\State if \( \alpha \) is too large \( \rightarrow \) return \ // \text{for the next round}
\EndIf
\EndFor
\State return \( \gamma \)
\end{algorithmic}
\end{algorithm}

It is worth mentioning that \( \alpha \) used in line 6 of Algorithm 2 is a tuning parameter whose value is a natural number. It is introduced to accelerate the expance of \( LVG \). As demonstrated by the experimental results, a small value of \( \alpha \) reduces the \( LVG \) size but incurs high obstructed distance computation overhead, whereas a larger value of \( \alpha \) decreases the cost of obstructed distance calculation but increases the size of \( LVG \). Nonetheless, how to select the appropriate values of \( \alpha \) 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 \( LVG = \{p, p_1\} \), \( \gamma = 0 \), \( S_{\gamma_0} = \{q\} \), and \( S_{\gamma_0} = \emptyset \). First, ODC obtains \( |p_1, q| = \text{dist}(p_1, q) \) due to \( S_{\gamma_0} = \{q\} \). As \( |p_1, q| > \gamma = 0 \), the algorithm performs while-loop in lines 5–18 of Algorithm 2, in which it sets \( \gamma \) to \( \text{dist}(p_1, q) \), calls GetObs to get \( S_{\gamma_0} = \{o_1, o_2\} \) and \( S_{\gamma_0} = \{a, b, c, d, e, f, g, h\} \), 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 \( S_{\gamma_0} \), 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 \( S_{\gamma_0} = \emptyset \). Then, the algorithm proceeds in the similar manner until \( |p_1, q| \leq \gamma \) holds, after which \( LVG = \{q, p_1, a, b, c, d, e, f, g, h, i, j, k, l\} \), \( \gamma = ||p_1, q|| = \text{dist}(p_1, a) + \text{dist}(a, d) + \text{dist}(d, q) \), \( S_{\gamma_0} = \{q, a, b, c, d, e, f, g, h, i, j, k, l\} \), finally, vertex \( i \) is also added to \( S_{\gamma_0} \) because \( |i, q| = \text{dist}(i, q) < ||p_1, q|| \). Here, the algorithm terminates, with \( LVG = \{q, p_1, a, b, c, d, e, f, g, h, i, j, k, l\} \), \( \gamma = ||p_1, q|| \), \( S_{\gamma_0} = \{q, a, b, c, d, e, f, i, j\} \), and \( S_{\gamma_0} = \{g, h, j, k, l\} \).

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 \( \gamma_p \) 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 \( \gamma \), a boundary region set \( S_{\text{BR}} \)

**Output:** a boundary region set \( S_{\text{BR}} \)

1. initialize \( V_p = \emptyset \)
2. Dijkstra\((LVG, p)\)
3. for each vertex \( v \in LVG \) do
4. if \([v, q] \leq \gamma \) and \([p, v] < [v, q] \) and \([p, v] < [v, q] \) then
5. \( V_p = V_p \cup \{v\} \) if \( v \) is a boundary vertex w.r.t. \( q \)
6. if \( V_p \neq \emptyset \) then
7. find \( v_{\text{min}} \) and \( v_{\text{max}} \) from \( V_p \), and obtain \( BAp \) based on Definition 4.3
8. \( d_p = \max \{SP(p, v) \mid v \in \text{current } LVG \} \) dist\((v', q)\)
9. \( S_{\text{BR}} = S_{\text{BR}} \cup \{p, BAp, d_p, SP(p, v_{\text{min}}), SP(p, v_{\text{max}})\} \)
10. return \( S_{\text{BR}} \)

**Fig. 5.** Example of BRF.

It is worth noting that \( P(p, q) \) is updated when \( LVG \) expands. According to Lemma 5.2, the minimal \( P(p, q) \) 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 vertices 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 vertices. 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 \( \langle p, BAp, d_p, SP(p, v_{\text{min}}), SP(p, v_{\text{max}}) \rangle \) (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 \( LVG = \{\{q, p, a, b, c, d, e, f, g, h, i, j, k, l\} \), \( S_{\text{BR}} = \emptyset \), and \( \gamma \). 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 \( v_{\text{min}} = f \) and
Algorithm 4
ORN NN Search Algorithm (ORN NN).

**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

1. initialize \( S_r = S_v = \emptyset \) and \( LVG = [q] \)
2. \( S_r \leftarrow \text{ORN NN-Filter}(T_p, T_o, q, S_r, LVG) \) // Algorithm 5
3. \( S_r \leftarrow \text{ORN NN-Refinement}(T_p, T_o, q, S_r, S_v) \) // Algorithm 6
4. return \( S_r \)

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

**Input:** a data R-tree \( T_p \), an obstacle R-tree \( T_o \), a query point \( q \), a candidate set \( S_r \), a local visibility graph \( LVG \)

**Output:** a candidate set \( S_r \)

\( \because H_p, H_o: \) min-heaps; \( T_p . \text{root}: \) the root of \( T_p \); \( T_o . \text{root}: \) the root of \( T_o \)

1. initialize \( \gamma = 0, S_v = S_v = \emptyset, S_v \leftarrow [q] \)
2. initialize \( H_p = (T_p . \text{root}, 0), H_o = (T_o . \text{root}, 0) \)
3. while \( H_p \) is not empty do
4. de-heap the top entry \((e, \text{mindist}(e, q))\) of \( H_p \)
5. if \( e \) is a data point \( p \) then
6. if not \( \text{BRP}(p, q, S_v) \) then // Algorithm 1
7. add \( p \) to \( LVG \)
8. \( \gamma \leftarrow \text{ODC}(p, q, LVG, \gamma, T_v, H_p, S_v, S_v) \) // Algorithm 2
9. \( S_v = S_v \cup \{p\} \)
10. \( S_v \leftarrow \text{BRF}(p, q, LVG, \gamma, S_v) \) // 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 \( \text{BRP}(e_i, q, S_v) \) then
15. insert \((e_i, \text{mindist}(e_i, q))\) into \( H_p \)
16. if approximation is allowed then
17. if \( \text{LVG} \leftarrow \text{BA}\) = \( 2\pi \) and \( \text{MAX}_{p \in S_v} d_p \leq \text{mindist}(e, q) \) then
18. break // terminate algorithm
19. return \( S_r \)

\( v_{max} = d \), gets \( p \)’s boundary angle \( \text{BA}_p = \text{\angle fqd} \), derives \( d_p = \text{dist}(p, q) \), and updates the boundary region set \( S_{BR} \) to \((p, \text{\angle fqd}, \text{dist}(p, q), (p, f), (p, c, d))\).

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 (ORN NN). 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 \( S_r \) 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., \( \text{mindist} \)) 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 (ORN NN-Filter) is presented in Algorithm 5.

First, ORNN-Filter sets the radius \( \gamma \) 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 \( \text{BA}_p \) w.r.t. \( q \). 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., \( \cup_{p \in S_{BR}} \text{BA}_p = 2\pi \)), the search space for ORNN objects is restricted to a circle \( \text{cir}(q, d) \) centered at \( q \) and having \( d = \text{MAX}_{p \in S_{BR}} d_p \) as its radius. Thus, once the key (i.e., \( \text{mindist}(e, q) \)) 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 be effective.
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 \( d = \max_{p \in S_{BR}} d_p \), 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) \( \cup_{p \in S_{BBR}} B\mathcal{P}_e = 2\pi \).

**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 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 \( H_B = (N_5, N_6) \) sorted in ascending order of their mindist to a given query point \( q \). Then, the algorithm de-heaps the top entry \( N_5 \) of \( H_B \), accesses its child nodes, and en-heaps the entries into \( H_B = \{N_1, N_6, N_2\} \). Next, \( N_1 \) is visited and it updates \( H_B \) to \( \{p_1, N_6, p_2, N_2\} \). Since point \( p_1 \) is the head entry of \( H_B \) and cannot be discarded by BRP (due to \( S_{BR} = \emptyset \)), ORNN-Filter adds \( p_1 \) to a local visibility graph \( \mathcal{L}VG \), calls \( \mathcal{O}DC \) to calculate the obstructed distance from \( p_1 \) to \( q \), inserts \( p_1 \) into \( S_e \) (\( = \{p_1\}\) ), utilizes \( \mathcal{B}RF \) to determine \( p_1 \)'s boundary region \( B\mathcal{P}_{p1} \) i.e., polygon \( \mathcal{B}QP_{p1} a \) w.r.t. \( q \), updates \( S_{BR} \) to \( \{p_1, \mathcal{B}QP_{p1} a, \text{dist}(b, q), p_1, b, \{p_1, a\}\} \), and deletes \( p_1 \) from \( \mathcal{L}VG \). The algorithm proceeds in the same manner until the heap \( H_B \) becomes empty, with \( S_e = \{p_1, p_2\} \). Note that, the de-heaped entries from \( H_B \) (including \( p_2, N_2, p_3, p_4 \), and \( N_4 \)) during the search are pruned by \( B\mathcal{P}_{p1} \) or \( B\mathcal{P}_{p2} \).

### 6.2. The refinement step

Once the candidate set \( S_e \) 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 \( S_e \) \( = \{p_1, p_3\} \) after the termination of ORNN-Filter. Now we invoke ORNN-Refinement to verify each candidate in \( S_e \). 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.

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

**Proof.** The ORNN algorithm follows the filter-refinement framework, and \( |\mathcal{L}VG| \ll |O| \) (as demonstrated by the experimental results to be presented in **Section 8**). In the filtering step, it takes \( O(|S_e| \times \log|P| \times |O| \times \log|O|) \) for obtaining the candidate set \( S_c \); and in the refinement step, it incurs \( O(|S_e| \times C_{\text{conn}}) \) to eliminate all the false hits. Thus, the total time complexity of the ORNN algorithm is \( O(|S_e| \times \log|P| \times |O| \times \log|O| + C_{\text{conn}}) \). \( \Box \)
It is worth mentioning that, the time complexity of ORNN algorithm, i.e., $O(|S_c| \times (\log|P| \times |O| \times \log|O| + C_{\text{orn}}))$, is scalable. This is because $O(|S_c| \times (\log|P| \times |O| \times \log|O| + C_{\text{orn}}))$ is log scalable and linear scalable with respect to $|P|$ and $|O|$, respectively. Moreover, the time complexity $O(|S_c| \times (\log|P| \times |O| \times \log|O| + C_{\text{orn}}))$ of ORNN algorithm is better than that of the naive ORNN algorithm $O(|P| \times C_{\text{orn}})$, 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 $C_{\text{orn}} = O(|P| \times |O|)$. Hence, $O(|S_c| \times (\log|P| \times |O| \times \log|O| + C_{\text{orn}})) = O(|S_c| \times (\log|P| \times |O| \times \log|O| + |P| \times |O|))$ and $O(|P| \times C_{\text{orn}}) = O(|P| \times |P| \times |O|)$. As demonstrated by the experimental results, $|S_c| < |P|$. Thus, $O(|S_c| \times (\log|P| \times |O| \times \log|O| + C_{\text{orn}}))$ is better than $O(|P| \times C_{\text{orn}})$, 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., OR$k$NN, $\delta$-OR$k$NN, and COR$k$NN queries.

7.1. OR$k$NN search

As defined in **Definition 7.2**, an OR$k$NN 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 $\text{OkNN}(p)$, iff there are at most $k - 1$ points $r$ in $P$ satisfying $||r,p|| \leq ||p',p||$.

**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 (OKRNN) query retrieves a set $\text{OkRNN}(q) \subseteq P$ of points that have $q$ as one of their OkNNs, i.e., $\text{OkRNN}(q) = \{p \in P \mid q \in \text{OkNN}(p)\}$.

Take Fig. 1b as an example, and suppose $k = 2$. Since $O2 \text{NN}(p_1) = \{p_2, q\}$, $O2 \text{NN}(p_2) = \{q, p_3\}$, and $O2 \text{NN}(p_3) = \{q, p_2\}$, $O2 \text{NN}(q) = \{p_1, p_2, p_3\}$.

Next, we explain how to extend ORNN search algorithm to answer OR$k$NN 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 OR$k$NN retrieval, only when the MBR falls completely inside a pruning
To solve the OR
Constrained Obstructed Reverse
Definition 7.4.
the example. The manager may only be able to go certain distances to distribute coupons, e.g., not exceeding 2 km. He/she can use
Definition 7.3.
returns TRUE.

\[ \text{count} = \text{count} + 1 // \text{Lemma 4.1} \]

\[ \text{if} \ SP(p', q) \text{ crosses } SP(p, vmin) \text{ or } SP(p, vmax) \text{ then} \]

\[ \text{if} \ SP(p, q) \in S_Bk \text{ and dist}(p', q) > d_p \text{ then} // \text{Heuristics 1, 2} \]

\[ \text{if} \ e \text{ is a data point } p' \text{ or a node MBR } N \text{ then} \]

\[ \text{for each entry } (p, B_{Ap}, d_p, SP(p, vmin), SP(p, vmax)) \in S_B \text{ do} \]

\[ \text{if} \ \exists p, q \in \text{OkNN} \text{ such that } d_p \leq \text{mindist}(e, q) \text{, ORNN-Filter stops. However, the early} \]

\[ \text{else} // \text{exact ORNN search} \]

\[ \text{if} \ SP(p', q) \text{ crosses } SP(p, vmin) \text{ or } SP(p, vmax) \text{ then} \]

\[ \text{if} \ SP(p, q) \in S_Bk \text{ and dist}(p', q) > d_p \text{ then} // \text{Heuristics 1, 2} \]

\[ \text{if} \ e \text{ is a data point } p' \text{ or a node MBR } N \text{ then} \]

\[ \text{if} \ e \text{ is a data point } p' \text{ or a node MBR } N \text{ then} \]

\[ \text{for each entry } (p, B_{Ap}, d_p, SP(p, vmin), SP(p, vmax)) \in S_B \text{ do} \]

\[ \text{if} \ \exists p, q \in \text{OkNN} \text{ such that } d_p \leq \text{mindist}(e, q) \text{, ORNN-Filter stops. However, the early} \]

\[ \text{else} // \text{exact ORNN search} \]

\[ \text{if} \ SP(p', q) \text{ crosses } SP(p, vmin) \text{ or } SP(p, vmax) \text{ then} \]

\[ \text{return TRUE} // e \text{ can be pruned} \]

\[ \text{return FALSE} \]

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 \( S_Bk \) accepting entries in the form \( (p, B_{Ap}, d_p, SP(p, vmin), SP(p, vmax)) \), a parameter \( k \)

Output: TRUE if \( e \) can be pruned or FALSE otherwise

1: initialize \( \text{count} = 0 // \text{count: a counter} \)
2: if \( e \text{ is a data point } p' \text{ or a node MBR } N \text{ then} \)
3: for each entry \( (p, B_{Ap}, d_p, SP(p, vmin), SP(p, vmax)) \in S_B \text{ do} \)
4: if approximation is allowed then
5: if \( \exists p, q \in \text{OkNN} \text{ such that } d_p \leq \text{mindist}(e, q) \text{, ORNN-Filter stops. However, the early} \)
6: \( \text{count} = \text{count} + 1 // \text{Lemma 4.1} \)
7: else // exact ORNN search
8: if \( SP(p', q) \text{ crosses } SP(p, vmin) \text{ or } SP(p, vmax) \text{ then} \)
9: \( \text{count} = \text{count} + 1 // \text{Lemma 4.1} \)
10: if \( \text{count} \geq k \text{ then} \)
11: return TRUE // \( e \text{ can be pruned} \)
12: return FALSE

Fig. 8. Example of \( \delta \text{-ORNN} \) 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 \( S_{RB} \) 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 \( S_{RB} \) (i.e., \( \max_{p \in S_{RB}} d_p \leq \text{mindist}(e, q) \)), ORNN-Filter stops. However, the early termination condition for ORNN-Filter is strict. Only when the existing boundary regions stored in \( S_{RB} \) 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 \( S_{RB} \). 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 \( \delta \) 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 \( \delta \)). Given a data set \( P \), an obstacle set \( O \), a query point \( q \), and an obstructed distance threshold \( \delta \), an ORkNN query with maximum obstructed distance \( \delta \) (\( \delta \text{-ORkNN} \)) returns a set \( \delta \text{-ORkNN}(q) \subseteq P \) points, such that \( \forall p \in \delta \text{-ORkNN}(q), q \in \text{OkNN}(p) \text{ and } ||p, q|| \leq \delta, \text{i.e., } \delta \text{-ORkNN}(q) = \{p \in P \mid q \in \text{OkNN}(p) \land ||p, q|| \leq \delta\} \).

An example of \( \delta \text{-ORNN} (k = 1) \) search is depicted in Fig. 8a, where \( p_2 \) is not the \( \delta \text{-ORNN} \) of \( q \) due to \( ||p_2, q|| > \delta \), while \( p_3 \) is a \( \delta \text{-ORNN} \) of \( q \) as \( ||p_3, q|| < \delta \) and \( \text{ONN}(p_2) = \{q\} \). The \( \delta \text{-ORNN} \) 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 \( \delta \text{-ORNN} \) 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 \( \text{CORkNN}(q) \subseteq P \) points, such that \( \forall p \in \text{CORkNN}(q), q \in \text{OkNN}(p) \text{ and } p \cap CR \neq \emptyset, \text{i.e., } \text{CORkNN}(q) = \{p \in P \mid q \in \text{OkNN}(p) \land p \cap CR \neq \emptyset\} \).
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 support decision support.

The proposed algorithms for ORNN search are flexible, and can be naturally adjusted to support \(\delta\)-ORNN and CORNN queries, by integrating constrained conditions (i.e., the distance threshold \(\delta\) 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 \(\delta\)-ORNN retrieval is bounded by \(\delta\) (e.g., the shaded area in Fig. 8a represents the SR of the \(\delta\)-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 \(\text{mindist}\) to \(q\) (i.e., \(\text{mindist}(e, q)\) larger than \(\delta\), 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 \(\delta\) 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 \(\delta\)-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, \(SC\) 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 \(S_\ell\) (i.e., \(|S_\ell|\)), 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.

<table>
<thead>
<tr>
<th>Table 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Real datasets used in experiments.</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter ranges and default values.</td>
</tr>
<tr>
<td>Parameter</td>
</tr>
<tr>
<td>(\alpha)</td>
</tr>
<tr>
<td>(k)</td>
</tr>
<tr>
<td>(\left</td>
</tr>
<tr>
<td>buffer size (% of the tree size)</td>
</tr>
<tr>
<td>(\delta) (% of the space width)</td>
</tr>
<tr>
<td>(\alpha) (% of full space)</td>
</tr>
</tbody>
</table>
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 OR$k$NN algorithms employing different heuristics. Fig. 9 plots the efficiency of different heuristics with respect to $\alpha$, $|P|/|O|$, $k$, and buffer size respectively, using dataset combinations $SG$ and $SL$. Specifically, Naive denotes the OR$k$NN algorithm without any heuristic; H12 represents the OR$k$NN algorithm using Heuristics 1 and 2; and H3 denotes the OR$k$NN 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 OR$k$NN 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 OR$k$NN queries. First, we investigate the effect of $\alpha$ on the presented OR$k$NN 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.
Notice that, the cost of OR$k$NN search drops with $\alpha$. The reason is that, as $\alpha$ becomes larger, the number of obstructed distance computation decreases significantly and the boundary region formation speeds up. Also notice that, although $|LVG|$ increases with $\alpha$, it 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 $\alpha$. It is observed that the PH ratio decreases slightly as $\alpha$ grows. The reason is that when $\alpha$ 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 $\alpha$ 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 $\alpha$ 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. OR\(k\)NN search performance vs. buffer size (\(\alpha = 4, |P|/|O| = 1, k = 5\)).

Fig. 14. \(\delta\)-OR\(k\)NN search performance vs. \(\delta\) (\(\alpha = 4, |P|/|O| = 1, k = 5\)).

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 OR\(k\)NN-Refinement improves although the cost of OR\(k\)NN-Filter remains. This is because, the refinement step of OR\(k\)NN search requires performing an OR\(k\)NN 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 OR\(k\)NN queries with constraints

The last set of experiments evaluates the performance of algorithms for \(\delta\)-OR\(k\)NN and COR\(k\)NN queries.

First, we inspect the influence of the maximum obstructed distance \(\delta\) on the efficiency of \(\delta\)-OR\(k\)NN search algorithm. We change \(\delta\) 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, \(\delta\) has a direct impact on the performance of \(\delta\)-OR\(k\)NN retrieval, since it controls the size of the search region. In particular, the cost of the algorithm increases gradually as \(\delta\) grows. The reason is that with the growth of \(\delta\), 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 COR\(k\)NN 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.
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., OR\(k\)NN, \(\delta\)-OR\(k\)NN, and COR\(k\)NN queries, and conduct extensive experiments with both real and synthetic datasets to demonstrate the effectiveness of our proposed pruning heuristics and the performance of our proposed algorithms. In the future, we intend to explore the OR\(k\)NN 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


