Continuous Obstructed Detour Queries

Rudra Ranajee Saha
Department of CSE, Bangladesh University of Engineering and Technology, Dhaka, Bangladesh
darklord.saha@gmail.com

Tanzima Hashem
Department of CSE, Bangladesh University of Engineering and Technology, Dhaka, Bangladesh
tanzimahashem@cse.buet.ac.bd

Tasmia Shahriar
Department of CSE, Bangladesh University of Engineering and Technology, Dhaka, Bangladesh
shahriartasmia@gmail.com

Lars Kulik
Dept of CIS, University of Melbourne, Melbourne, Australia
lkulik@unimelb.edu.au

Abstract

In this paper, we introduce Continuous Obstructed Detour (COD) Queries, a novel query type in spatial databases. COD queries continuously return the nearest point of interests (POIs) such as a restaurant, an ATM machine and a pharmacy with respect to the current location and the fixed destination of a moving pedestrian in presence of obstacles like a fence, a lake or a private building. The path towards a destination is typically not predetermined and the nearest POIs can change over time with the change of a pedestrian's current location towards a fixed destination. The distance to a POI is measured as the summation of the obstructed distance from the pedestrian's current location to the POI and the obstructed distance from the POI to the pedestrian's destination. Evaluating the query for every change of a pedestrian's location would incur extremely high processing overhead. We develop an efficient solution for COD queries and verify the effectiveness and efficiency of our solution in experiments.

2012 ACM Subject Classification Information systems → Location based services

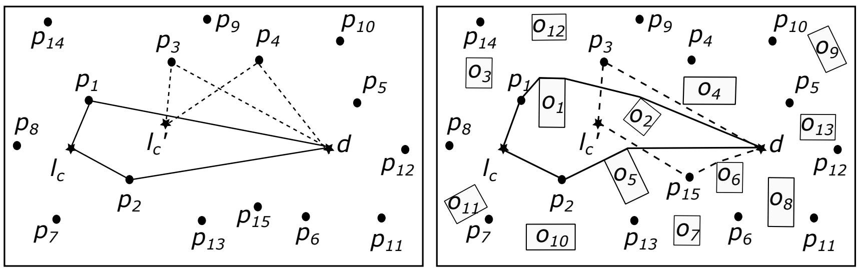
Keywords and phrases Obstacles Continuous Detour Queries Spatial Databases

Digital Object Identifier 10.4230/LIPIcs.GIScience.2018.14

Acknowledgements I This research was partially supported under the Australian Research Council's Discovery Projects funding scheme (project number DP180103332).

1 Introduction

Efficient processing of location-based queries in the presence obstacles like a river, a fence or a private property has become an important research area in recent years. Obstructed space is different from road networks and the Euclidean space, which ignore the obstacles in the space. It is not possible to adapt the query processing algorithms for the Euclidean space or road network settings to the obstructed space as the presence of obstacles brings new challenges for processing location-based queries in real time. Considering the importance of the applications of obstructed location-based queries for pedestrians, in the last few years, researchers have developed solutions [1, 5, 16, 20] for variant location-based queries in the obstructed space that were previously addressed in the Euclidean space or road networks.



(a) Euclidean Space. **(b)** Obstructed Space.

Figure 1 An Example of a Continuous Detour Query for $k = 2$.

We introduce a Continuous Obstructed Detour (COD) query that allows a moving pedestrian continuously monitor the POI with the smallest obstructed detour distance, which is measured as the summation of distances from the user's current location to the POI, and from the POI to the user's destination by avoiding the obstacles. For example, a tourist enjoying a scenic view may not follow a predetermined walking path and instead want to visit a restaurant or a souvenir shop before arriving at the hotel. A pedestrian roaming around the city may want to buy a medicine from a pharmacy before she goes to her usual bus stop to home. In both scenarios, users have fixed destinations but do not have a predetermined path to reach the destination, and need to visit a POI before reaching the destinations.

A COD query can be extended to a COkD query that continuously returns k POIs with the k smallest detour distances for a moving user heading towards a fixed destination. Figure 1 shows an example of a continuous detour query for $k = 2$ in both Euclidean and obstructed space. The Euclidean distance is measured as the length of the direct line connecting two locations. In Figure 1(a), when a user is at l_c, POIs p_1 and p_2 are the 1st and 2nd nearest detour POIs based on the Euclidean distances. When the user moves to $l_c{}'$, the answer changes, and p_3 and p_4 become the 1st and 2nd nearest detour POIs based on the Euclidean distances. In Figure 1(b), the obstacles are shown using rectangles. The obstructed distance is the length of the shortest path between two locations without crossing any obstacle. Figure 1(b) shows that p_2 and p_1 are the 1st and 2nd obstructed nearest detour POIs when the user is at l_c. When the user moves to $l_c{}'$, the answer changes, and p_{15} and p_3 become the 1st and 2nd obstructed nearest detour POIs.

The COkD query cannot be modeled and processed as a continuous obstructed nearest neighbor (POI) query because of the presence of a destination. Although the obstructed distance of a POI to the destination is constant, it differs for multiple POIs, and the obstructed nearest detour POI is determined with respect to both current location and destination of the moving pedestrian. Hence, the solution [10] for moving k nearest neighbor (kNN) queries in the obstructed space is not applicable for COkD queries.

Since the path to reach the destination is not predefined, for a COkD query, the obstructed nearest detour POIs need to be re-evaluated in real time with respect to every changed location and the destination location of the moving user. Thus, a COkD query can be processed with the repeated evaluation of obstructed k detour (OkD) queries, where an OkD query returns k obstructed nearest detour POIs with respect to a user's current location and destination. Researchers have proposed obstructed k group nearest neighbor (OkGNN) algorithms [15, 16] that return k POIs having k smallest obstructed aggregate distances with respect to multiple query locations. An OkGNN query is same as an obstructed k detour (OkD) query when the number of query location is two. However, the straightforward application of the OkD algorithm for processing a COkD query is not feasible as it would incur extremely high processing overhead, specially in the obstructed space as the computation of the number of obstructed distance increases with the increase of the number of the query re-evaluation. The search for the obstructed nearest detour POIs independently using OkD queries accesses the same POIs and obstacles multiple times. Thus, the major challenges for processing a COkD query efficiently is to reduce the frequency of the query re-evaluation and the retrieval of the same POIs and obstacles from the database.

To address the challenges for a COkD query, we develop a safe region [10, 13] based solution that avoids the re-evaluation of the query as much as possible. The key idea of our safe region based approach is to retrieve the obstructed nearest detour POIs from a database with respect to a user's current location and destination, and then identify the regions, obstructed integrated safe region (OISR) and obstructed safe regions (OSRs) with respect to the retrieved POIs. We exploit geometric properties to compute such regions. If a user resides in the OISR, the user's movement does not change the order of already retrieved k obstructed nearest detour POIs. Thus, the computation of an OISR avoids the re-computation of the query answer. If a user leaves an OISR, we compute obstructed safe regions (OSRs) with respect to the retrieved POIs to check whether new POIs are required to be retrieved from the database. Computation of OSRs allows us to avoid the retrieval of the same POIs multiple times, which in turn decreases the number of same obstacles retrieved for computing the obstructed distances of the POIs.

To further improve the efficiency of our approach, we propose two algorithms: a single point retrieval method (SPRM) and a multiple point retrieval method (MPRM), to retrieve new POIs from the database when a moving user leaves the current safe region. The aim of SPRM and MPRM is to refine the POI search space, i.e., reduce the number of the retrieval of new POIs, for identifying k obstructed nearest detour POIs with respect to the current location l_c and destination d of a moving user. A smaller number of retrieved POIs reduces the computational overhead and I/O cost for retrieving obstacles from the database.

The key difference between SPRM and MPRM is that SPRM incrementally retrieves obstructed nearest detour POIs with respect to the first location and the destination of the moving user (e.g., l_c and d in Figure 1), whereas for MPRM the obstructed nearest detour POIs are retrieved with respect to few of the current locations and destination of the moving user (e.g., l_c and d, and $l_c{}'$ and d in Figure 1). SPRM does not retrieve the same POI multiple times but may retrieve additional POIs, whereas MPRM reduces the retrieval of additional POIs in return of increasing the number of obstructed distance computations with respect to multiple locations (e.g., l_c and $l_c{}'$ in Figure 1).

We summarize our key contributions below:

- We introduce and formulate COkD queries in spatial databases that allow pedestrians to monitor the nearest detour POIs in the presence of obstacles.
- We develop an efficient safe-region based solution for processing COkD queries. To the best of our knowledge, we are the first to address the problem of COkD queries.
- We develop two algorithms, SPRM and MPRM, to refine the POI search space and retrieve new POIs in the refined search space.
- We perform extensive experiments using a real data set to show the efficiency and effectiveness of our proposed solution.

Table 1 A List of Symbols.

Notation	Description	Notation	Description
k	The number of required nearest detour POIs	x	The number of auxiliary POIs
l_c	The current location	d	The destination
l_f	The location from where a user starts to move	o_i	An obstacle
l_s	The location used to compute safe regions	O	The set of obstacles
z	The $(k+x)^{th}$ nearest POI of l_s	P	The set of all POIs
p_i	A POI	L	The list (set) of $(k+x)$ obstructed nearest detour POIs
A	The set of k obstructed nearest detour POIs for l_c and d	T_p	POI R-tree
$d_e(p,q)$	Euclidean distance between p and q	$d_\triangle(p,q)$	Obstructed distance between p and q
$s_e(a,b,c)$	Summation of $d_e(a,b)$ and $d_e(b,c)$	$s_{e\triangle}(a,b,c)$	Summation of $d_e(a,b)$ and $d_\triangle(b,c)$
$s_\triangle(a,b,c)$	Summation of $d_\triangle(a,b)$ and $d_\triangle(b,c)$	T_o	Obstacle R-tree

2 Problem Formulation

In a COkD query, initially, a moving user provides her current location l_c, a destination d and the number k of desired nearest (detour) POIs. Later the moving user periodically updates her current location l_c. The obstructed space may include obstacles like buildings, parks, lakes, etc. An obstructed path is calculated as the shortest path between two points in the obstructed space, where a path does not intersect the interior of an obstacle. The obstructed distance between two points is the length of the obstructed path between those points. The obstructed detour distance $s_\triangle(l_c, p_i, d)$ of a POI p_i is measured as the summation of the obstructed distances from p_i to l_c and p_i to d. Similar to existing work in the obstructed space [16, 15], the POIs and obstacles are indexed using two separate R-trees [7], POI R-tree and obstacle R-tree in the database. Table 1 summarizes the symbols used in the paper.

A COkD query is formally defined as follows.

▶ **Definition 1. A Continuous Obstructed *k* Detour Query:** Given a set of POIs P and a set of obstacles O, the current location l_c of a moving user, a destination d, and the required number of the obstructed nearest detour POIs k, a COkD query returns A, a set of k obstructed nearest detour POIs that have k smallest obstructed detour distances with respect to every instance of l_c and d, i.e., $s_\triangle(l_c, p_i, d) \leq s_\triangle(l_c, p_j, d)$ for $p_i \in A$ and $p_j \in P - A$.

3 Related Work

Efficient approaches have been proposed in the literature for variants of spatial queries in the obstructed space. Processing spatial queries in the presence of obstacles has been first addressed in [19]. In [6, 17, 19], the authors developed algorithms to find the nearest POIs with respect to a static location in the obstructed space. In [5], the authors developed an algorithm to process continuous obstructed nearest neighbor queries. In [4] and [1], the authors developed solutions for efficient processing of obstructed reverse nearest neighbor queries and obstructed optimal sequenced route queries, respectively. In [20], the authors addressed obstructed range nearest neighbor queries. Obstructed group nearest neighbor (OGNN) queries that return a POI with the minimum obstructed aggregate distance have been addressed in [15, 16]. An OGNN query transforms to an obstructed detour query if the number of query location is two (i.e., a user's current location and destination). This paper focuses on the COkD query, which is different from all of the above mentioned queries.

In [19], the authors proposed the first algorithm to compute the obstructed distance between two locations. Instead of directly applying the obstructed distance computation algorithm between two locations, in [16], the authors developed an algorithm to efficiently compute multiple obstructed distances with respect to a single point without retrieving same obstacles multiple times. To compute the obstructed detour distance, we need to compute two obstructed distances from a common POI, and thus, we use the algorithm in [16].

Continuous nearest neighbor queries [3, 11] and continuous detour queries [12, 14, 18] have been addressed in road networks that ignore the presence of the obstacles. In [12], the authors proposed an incremental approach using a shortest path tree to process continuous detour queries in the road network. In [14, 18], the authors developed a solution for detour queries with an assumption that a user travels in a predetermined path towards a destination. In COkD queries, a pedestrian's path towards a destination is not known before and can be obstructed by the obstacles.

Researchers have already shown that computing safe regions can significantly reduce the query processing overhead for processing moving nearest neighbor queries [8, 10, 13]. However, none of these approaches take the destination into account, and thus, the computed safe regions are not applicable for a COkD query, where a pedestrian moves towards a fixed destination.

4 Safe Regions

We develop a safe region based approach for processing COkD queries. The underlying idea is to identify the safe regions based on already retrieved POIs, *obstructed integrated safe region (OISR)* and the intersection of *obstructed safe regions (OSRs)*, where the query answer does not change and any new POI does not need to be retrieved from the database for a moving user, respectively. These regions can help us to reduce the computational overhead and the retrieval of same POIs multiple times from the database. The larger the safe regions, the smaller is the number of times POIs need to be retrieved from the database. Considering this issue, we retrieve auxiliary POIs in addition to the required number (k) of POIs with an intuition that additional POIs can reduce the processing overhead. The number of auxiliary POIs x is decided in experiments.

Suppose that L is a set of ordered $k + x$ obstructed nearest detour POIs that have been retrieved from the database with respect to a moving user's location l_s and a fixed destination d for $x \geq 0$. An OSR of a retrieved POI represents the area, where a user's movement cannot incur another POI that has not yet been retrieved from the database to have a smaller obstructed detour distance than the retrieved POI. Thus, additional POIs are not retrieved from the database if a user moves inside the intersection of the OSRs of the retrieved POIs. An OISR represents an area where the current COkD answer for a moving user does not change. To compute the OISR, in addition to OSRs we need to know the obstructed fixed rank region (OFRR) that represents the area where a user's movement does not change the relative ranking (based on the obstructed detour distance) of the retrieved POIs in L.

In Sections 4.1 and 4.2, we show how the presence of a fixed destination d makes the computation of OSRs and OFRR different from the existing OSR and OFRR computation techniques for obstructed nearest neighbor queries [10]. In Section 4.3, we combine OSRs and OFRR to compute an OISR.

4.1 Obstructed Safe Region (OSR)

Let z represent the POI that has the $(k + x)^{th}$ smallest obstructed detour distance with respect to l_s and d. Based on the retrieved $k + x$ obstructed nearest POIs with respect to l_s and d, we first define the obstructed known region: a set of points that have equal or smaller obstructed detour distances than that of z with respect to l_s and d. Figure 2(a) shows an obstructed known region for $k = 2$ and $x = 1$, where p_1, p_2, and p_3 have been retrieved as $k + x$ obstructed nearest detour POIs with respect to l_f and d. Note that l_f is the location from where a user starts to move, and l_s is the location used to compute safe regions. Thus,

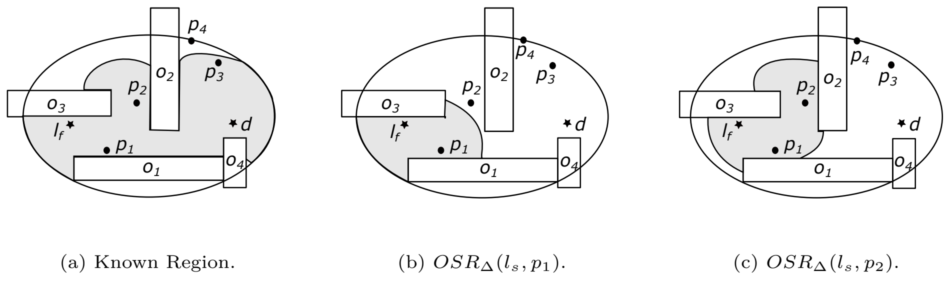


Figure 2 (a) Known Region, and (b)-(c) OSRs.

the first time when a safe region is computed, both l_f and l_s point to the same location. In all figures, we only show l_f and we assume that the safe regions are computed for the first time and l_s points to l_f.

Let p_o be a POI located outside the obstructed known region that has not yet been retrieved from the database. For a POI p_i in the obstructed known region, the obstructed safe region with respect to p_i, denoted by $OSR_\triangle(l_s, p_i)$, is defined as follows:

$$\begin{aligned} &OSR_\triangle(l_s, p_i) = \{l | s_\triangle(l, p_i, d) \leq s_\triangle(l, p_o, d)\} \\ &= \{l | d_\triangle(l, p_i) + d_\triangle(p_i, d) \leq d_\triangle(l, p_o) + d_\triangle(p_o, d)\} \end{aligned} \tag{1}$$

Here l refers to a point location. Thus $OSR_\triangle(l_s, p_i)$ is a set of points, where each point l satisfies $s_\triangle(l, p_i, d) \leq s_\triangle(l, p_o, d)$.

From the definition of the known region, $d_\triangle(l_s, p_o) + d_\triangle(p_o, d) \geq d_\triangle(l_s, z) + d_\triangle(z, d)$. Rearranging we have, $d_\triangle(l_s, p_o) \geq d_\triangle(l_s, z) + d_\triangle(z, d) - d_\triangle(p_o, d)$. On the other hand, according to the triangular inequality, $d_\triangle(l_s, l) + d_\triangle(l, p_o) \geq d_\triangle(l_s, p_o)$. By rearranging and replacing $d_\triangle(l_s, p_o)$ with its tighter bound, we have the tighter bound of $d_\triangle(l, p_o)$ as $d_\triangle(l_s, z) + d_\triangle(z, d) - d_\triangle(p_o, d) - d_\triangle(l_s, l)$.

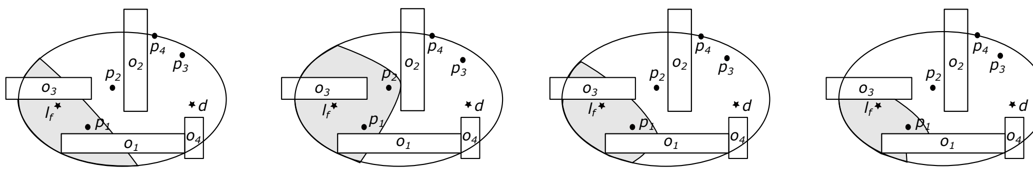
In Equation 1, if we can guarantee that $(d_\triangle(l, p_i) + d_\triangle(p_i, d))$ is less than or equal to a tighter bound of $(d_\triangle(l, p_o) + d_\triangle(p_o, d))$, i.e., $d_\triangle(l_s, z) + d_\triangle(z, d) - d_\triangle(l_s, l)$, then $d_\triangle(l, p_i) + d_\triangle(p_i, d) \leq d_\triangle(l, p_o) + d_\triangle(p_o, d)$ is satisfied. Thus, we can redefine $OSR_\triangle(l_s, p_i)$ as follows:

$$\begin{aligned} &OSR_\triangle(l_s, p_i) \\ &= \{l | d_\triangle(l, p_i) + d_\triangle(p_i, d) \leq d_\triangle(l_s, z) + d_\triangle(z, d) - d_\triangle(l_s, l)\} \\ &= \{l | d_\triangle(l, p_i) + d_\triangle(l_s, l) \leq d_\triangle(l_s, z) + d_\triangle(z, d) - d_\triangle(p_i, d)\} \end{aligned} \tag{2}$$

Figures 2(b) and 2(c) show OSRs for p_1 and p_2, respectively for the same example shown in Figure 2(a). According to Equation 2, if a moving user's current location l_c satisfies $d_\triangle(l_s, l_c) + d_\triangle(l_c, p_i) \leq d_\triangle(l_s, z) + d_\triangle(z, d) - d_\triangle(p_i, d)$, then the user is inside the OSR of p_i, $OSR_\triangle(l_s, p_i)$, and any POI p_o outside the obstructed known region cannot have a detour distance smaller than that of p_i with respect to l_c and d. If the user's current location l_c is inside the intersection of the OSRs for all $(k + x)$ POIs in the obstructed known region, i.e., $\bigcap_{i=1}^{k+x} OSR_\triangle(l_s, p_i)$, then it is guaranteed that any POI p_o outside the obstructed known region cannot have a detour distance smaller than those for $(k + x)$ POIs in the obstructed known region with respect to l_c and d.

4.2 Obstructed Fixed Rank Region (OFRR)

The OFRR represents an area where the ranking of k obstructed nearest detour POIs in L does not change. We compute an OFRR using the concept of a dominant region. In [10],



(a) $\beta_{e\Delta}'(p_1, p_2)$. (b) $\beta_{e\Delta}'(p_2, p_3)$. (c) $AOFRR(L)$. (d) $OFRR(L)$.

Figure 3 (a)-(b) Dominant Regions, (b) Approximate OFRR (c), and (d) OFRR (Shaded Areas).

for a moving obstructed nearest POI query, an obstructed dominant region of POI p_i over POI p_j is defined as $\beta_\Delta(p_i, p_j) = \{l | d_\Delta(l, p_i) <= d_\Delta(l, p_j)\}$. We modify the definition of a dominant region for a COkD query as follows:

$$\beta_\Delta(p_i, p_j) = \{l | s_\Delta(l, p_i, d) <= s_\Delta(l, p_j, d)\} \tag{3}$$

For a COkD query, an OFRR for an ordered POI set L can be computed as follows:

$$OFRR(L) = \bigcap_{i=1}^{|L|-1} \beta_\Delta(p_i, p_{i+1}) \tag{4}$$

To reduce the complexity of the computation of OFRRs, we first approximate a dominant region of POI p_i over POI p_j as $\beta_{e\Delta}'(p_i, p_j) = \{l | s_{e\Delta}(l, p_i, d) <= s_{e\Delta}(l, p_j, d)\}$.

Using the approximate dominant regions, we compute the approximate OFRR (AOFRR) for L as follows:

$$AOFRR(L) = \bigcap_{i=1}^{|L|-1} \beta_{e\Delta}'(p_i, p_{i+1}) \tag{5}$$

We continue with the same example shown in Figure 2(a). Figures 3(a) and 3(b) show the approximate dominant region of p_1 over p_2, $\beta_{e\Delta}'(p_1, p_2)$ and the approximate dominant region of p_2 over p_3, $\beta_{e\Delta}'(p_2, p_3)$, respectively.

After computing the $AOFRR(L)$ using Equation 5, we identify the non visible region inside $AOFRR(L)$ for every POI in L. Let NVR_i be a non visible region for a POI p_i and NVR be the union of non visible regions with respect to all POIs in L. Thus, an OFRR for an ordered POI set L can be computed from the approximated OFRR as follows:

$$OFRR(L) = AOFRR(L) - NVR \tag{6}$$

Figures 3(c) and 3(d) show $AOFRR(L)$ and $OFRR(L)$, respectively, where $L = \{p_1, p_2, p_3\}$. $AOFRR(L)$ (shaded area) in Figure 3(c) is computed as the intersection areas between the shaded areas, $\beta_{e\Delta}'(p_1, p_2)$ and $\beta_{e\Delta}'(p_2, p_3)$, in Figures 3(a) and 3(b), respectively. $OFRR(L)$ in Figure 3(d) is computed by removing the nonvisible regions of p_1, p_2, and p_3 from $AOFRR(L)$, i.e., $OFRR(L) \subseteq AOFRR(L)$.

4.3 Obstructed Integrated Safe Region (OISR)

The obstructed integrated safe region, denoted by OISR, is the area, where a user's movement does not change the COkD query answer. It is the intersection of OSR and OFRR. Formally the OISR can be defined as follows:

$$OISR(l_s, L) = OFRR(L) \cap \bigcap_{i=1}^{k} OSR_\Delta(l_s, p_i) \tag{7}$$

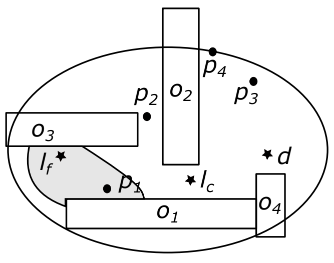


Figure 4 OISR.

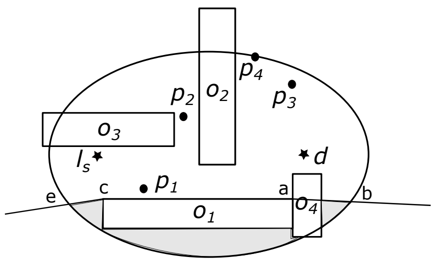


Figure 5 Non Visible Region for p_1.

The shaded area in Figure 4 shows the OISR for the same example shown in Figure 2(a). However, computing intersections of safe regions for every POI is expensive. The following theorem shows that the intersection of $OFRR(L)$ and $OSR_\Delta(l_s, p_k)$ is enough to generate $OISR(l_s, L)$.

▶ **Theorem 2.** *Given a set of retrieved ordered POIs L with respect to a moving user's locations l_s and d, the obstructed safe region $OSR_\Delta(l_s, p_i)$ for every i^{th} nearest POI p_i of l_s in L, the obstructed fixed rank region $OFRR(L)$, then $OFRR(L) \cap \bigcap_{i=1}^{k} OSR_\Delta(l_s, p_i) = OFRR(L) \cap OSR_\Delta(l_s, p_k)$.*

Proof. Suppose l_c is a location in $OFRR(L) \cap \bigcap_{i=1}^{k} OSR_\Delta(l_s, p_i)$. Since l_c is a location inside $OFRR(L)$, for $i \in [1..k-1]$, the following equation also holds:

$$d_\Delta(l_c, p_i) + d_\Delta(p_i, d) \leq d_\Delta(l_c, p_k) + d_\Delta(p_k, d) \tag{8}$$

Since $l_c \in OSR_\Delta(l_s, p_i)$, from Equation 2, we have

$$d_\Delta(l_c, p_i) + d_\Delta(p_i, d) \leq d_\Delta(l_s, z) + d_\Delta(z, d) - d_\Delta(l_s, l_c) \tag{9}$$

Now if Equation 9 holds for location l_c and $i = k$, then according to Equation 8, Equation 9 also holds for l_c and $i \in [1..k-1]$. Thus, $OFRR(L) \cap \bigcap_{i=1}^{k} OSR_\Delta(l_s, p_i) = OFRR(L) \cap OSR_\Delta(l_s, p_k)$. ◀

5 Algorithms

In this section, we present our COkD query processing algorithm (Algorithm 1) using safe regions computed in Section 4. The input to the algorithms are a current location l_c, a destination d, the number of required nearest detour POIs k, and the number of auxiliary POIs x. The output of the algorithm is the set of k obstructed nearest detour POIs A. Both l_c of a moving user and A are updated periodically. The algorithm uses a priority queue Q_p and lists L and L' to process a COkD query. Q_p is used to store already accessed R-tree nodes and POIs. L is a set of ordered $k + x$ obstructed detour POIs with respect to l_c and d. The list L' includes POIs that are not in L but have been retrieved from the database for finding $k + x$ obstructed nearest detour POIs.

Algorithm 1 starts with initializing l_f and l_s as l_c, where l_f is a moving user's start location and l_s is a location used to compute the last safe regions. Then the algorithm retrieves $k + x$ obstructed nearest detour POIs with respect to l_f and d using the function $RetrievePOIs$ in L (Line 2), adds first k obstructed nearest detour POIs in L to A (Line 3), and sends A to the

Algorithm 1 COkD_Process.

```
Input: $l_c$, $d$, $k$, $x$
Output: $A$

1: $l_f, l_s \leftarrow l_c$
2: $L \leftarrow RetrievePOIs(l_f, d, k, x)$
3: $A \leftarrow FindAnswer(l_c, d, k)$
4: $Send(A)$
5: for every update of $l_c$ do
6:     $flagOISR, L \leftarrow CheckOISR(l_s, l_c, d, k, x, L)$
7:     if $flagOISR = 1$ then
8:         $Send(A)$
9:     else
10:        $flagOSR \leftarrow CheckOSRs(l_s, l_c, d, k, x, L)$
11:        if $flagOSR = 0$ then
12:            $L \leftarrow RetrieveNextPOIs(l_f, l_c, d, k, x, L)$
13:            $l_s \leftarrow l_c$
14:        end if
15:        $A \leftarrow FindAnswer(l_c, d, k)$
16:        $Send(A)$
17:    end if
18: end for
```

user (Line 4). The function $RetrievePOIs$ incrementally retrieves Euclidean nearest detour POIs with respect to l_c and d from the database until $k + x$ obstructed nearest detour POIs for l_c and d have been identified. After every update of the current location l_c, the algorithm checks whether the current location l_c is in $OISR(l_s, L)$ using the function $CheckOISR$. The function returns 1 if $l_c \in OISR$, 0 otherwise. The steps of the function $CheckOISR$ are discussed in Section 5.1.

If the function $CheckOISR$ returns 1 (i.e., $l_c \in OISR$), then Algorithm 1 sends A to the user without any further computation (Lines 7-8). On the other hand, if the function $CheckOISR$ returns 0 (i.e., $l_c \notin OISR$), then Algorithm 1 checks whether l_c is in the intersection of OSRs of POIs $\{p_1, p_2, \ldots, p_k\}$ in L using the function $CheckOSRs$ (Line 10). The function checks the condition stated in the last line of Equation in 2 to determine whether $l_c \in OSR(p_i)$ of a POI p_i. If the condition is false for the OSR of any POI in $\{p_1, p_2, \ldots, p_k\}$, the function returns 0. If the condition is true for all POIs, then the function returns 1, i.e., l is in the intersection of OSRs of POIs $\{p_1, p_2, \ldots, p_k\}$ in L.

If $flagOSR = 1$, then the algorithm does not need to retrieve any new POI. If $flagOSR = 0$, then the algorithm retrieves $k + x$ nearest detour POIs in L with respect to l_c and d using the function $RetrieveNextPOIs$ (Line 12). For the function $RetrieveNextPOIs$, we develop two efficient methods: SPRM (Section 5.2) and MPRM (Section 5.3) with the aim to minimize the number of the retrieval of POIs for finding $k + x$ nearest detour POIs with respect to l_c and d. After retrieving new POIs using the function $RetrieveNextPOIs$, l_s is updated as l_c (Line 13). Finally, Algorithm 1 adds first k obstructed nearest detour POIs in L to A from L and sends A to the user (Lines 15-16).

Algorithm 2 CheckOISR.

```
Input: $l_s, l_c, d, k, x, L$
Output: $flagOISR$ and $L$

 1: $NVR \leftarrow ComputeNVR(L, l_s, d)$
 2: if $l_c \in NVR$ then
 3:     return $0, L$
 4: end if
 5: $flag, L \leftarrow CheckPOIOrder(L, l_c, d)$
 6: if $flag = 1$ then
 7:     return $0, L$
 8: else
 9:     return $l_c \in OSR(p_k), L$
10: end if
```

5.1 Function *CheckOISR*

The steps of this function is shown in Algorithm 2. The inputs to the algorithm are l_s, l_c, d, k, x, and L. The outputs are a flag $flagOISR$ and the L. From Equation 7, we know that $OISR(l_s, L)$ is the intersection of $OFRR(L)$ and $OSR(p_k)$. The function first computes non visible region NVR as the union of non visible regions with respect to all POIs in L (Line 1). if the direct path between a location and a POI is obstructed then the location is non-visible from the POI. Thus any location of a non visible region for a POI does not have a direct path to that POI. Figure 5 shows an example of non visible region (represented with two lines ab and ce) for POI p_1 with respect to obstacle O_1 by ignoring the presence of other obstacles. Non visible regions can be computed using a visibility graph [2, 9]. The vertices of a visibility graph represent POIs and corner points of the obstacles, and there is an edge between two vertices if the direct path between those vertices is not obstructed. To reduce the computational overhead, after computing a non visible region NVR_i for a POI p_i, it is stored and reused in the query evaluation process unless any new obstacle is retrieved.

Since $OFRR(L)$ can be computed as $AOFRR(L) - NVR$ (Equation 6), if l_c in NVR then l_c is not in $OFRR(L)$. Again from Equation 7, $OISR(l_s, L) = OFRR(L) \cap OSR_{\triangle}(l_s, p_k)$. Thus, if l_c in NVR then l_c is also not in $OISR(l_s, L)$. In such a scenario, Algorithm 2 returns $flagOISR$ as 0 and L without any modification (Lines 2-4).

Otherwise, using the function *CheckPOIOrder*, Algorithm 2 computes obstructed detour distances of POIs in L with respect to l_c and d, and sorts the POIs in L, if the order of POIs based on computed obstructed detour distances changes (Line 5). If the order is changed, $flag$ is set to 1 and Algorithm 2 returns $flagOISR$ as 0 and updated L (Lines 6-7). Otherwise, Algorithm 2 checks whether $l_c \in OSR(p_k)$ using the condition stated in the last line of Equation in 2 and returns $flagOISR$ as 1 or 0 and L without any modification, if the condition stated in the last line of Equation in 2 is satisfied or not, respectively.

5.2 SPRM

POIs are indexed using an R-tree in the database. To identify $(k + x)$ obstructed nearest detour POIs for l_c and d, SPRM incrementally retrieves Euclidean detour POIs with respect to l_f and d, where from l_f the user starts to move towards a destination d. A priority queue Q_p stores already accessed R-tree nodes and POIs in order of the minimum Euclidean detour

Table 2 Experiment Settings.

Parameter	Range	Default value	Parameter	Range	Default value
k	1-20	10	x	1-20	12
Query Range R	500-3000 units	1500 units	$\|P\|/\|O\|$ Ratio	50-350	200

distances with respect to l_f and d. To avoid the retrieval of the same POIs multiple times and reduce I/O access, SPRM does not start the search for the POIs from the root node of the R-tree while incrementally retrieving the POIs with respect to l_f and d.

The POI search space that has been already traversed is an ellipse with foci at l_f and d and the major axis having the length equal to the Euclidean detour distance of the last retrieved POI with respect to l_f and d from Q_p. SPRM determines the current $(k+x)^{th}$ smallest obstructed detour distance of l_c and d based on the already retrieved POIs. The ellipse expands with the retrieval of new POIs from Q_p. With the retrieval of a new POI, SPRM updates the current $(k+x)^{th}$ smallest obstructed detour distance of l_c and d if it becomes smaller. The search ends when the minimum Euclidean detour distance of l_c and d from the boundary of the ellipse becomes greater than or equal to the current $(k+x)^{th}$ smallest obstructed detour distance of l_c and d.

5.3 MPRM

Since SPRM expands the POI search space (i.e., ellipse) with respect to fixed locations l_f and d, some retrieved POIs may never become part of the COkD answer with respect to the updated location l_c and d. To avoid the retrieval of those additional POIs, MPRM retrieves new POIs with respect to l_c and d instead of l_f and d. Similar to SPRM, MPRM does not start the search from the root of the POI R-tree node and reuses the already traversed nodes of the POI R-tree. However, MPRM incurs additional processing overhead for computing the minimum Euclidean detour distances with respect to l_c and d for the nodes/POIs stored in Q_p.

MPRM sorts the already retrieved POIs according to the obstructed detour distance with respect to l_c and d. Then MPRM resorts the elements in Qp based on their Euclidean detour distances with respect to l_c and d. The algorithm continues to retrieve the next Euclidean nearest detour POI p with respect to l_c and d from Q_p as long as the Euclidean detour distance of p with respect to l_c and d is smaller than the current $(k+x)^{th}$ smallest obstructed detour distance of l_c and d based on already retrieved POIs.

6 Experiments

We present the performance of our safe region based approach using both SPRM and MPRM and compare them with a naive approach (NA) that independently finds k obstructed nearest detour POIs for every location update of a moving user using the OkD algorithm proposed in [16] (please see Section 3) for details. We use both real and synthetic data sets. The total space is normalized into $10,000 \times 10,000$ square units. The real dataset of Germany consists of 36334 Minimum Bounding Rectangles (MBRs) of railway lines (rrlines) and 76999 MBRs of hypsography data (hypos). In this dataset, end points of hypos represent POIs, and rrlines are the obstacles. Though we use MBRs to represent obstacles, our approach is applicable for obstacles of any shape. We also use the real datasets of rivers and lakes in Greece as obstacles, and generate synthetic POIs using uniform random distribution. We denote the synthetic dataset (Greece dataset) by 'S' and Germany dataset by 'G'.

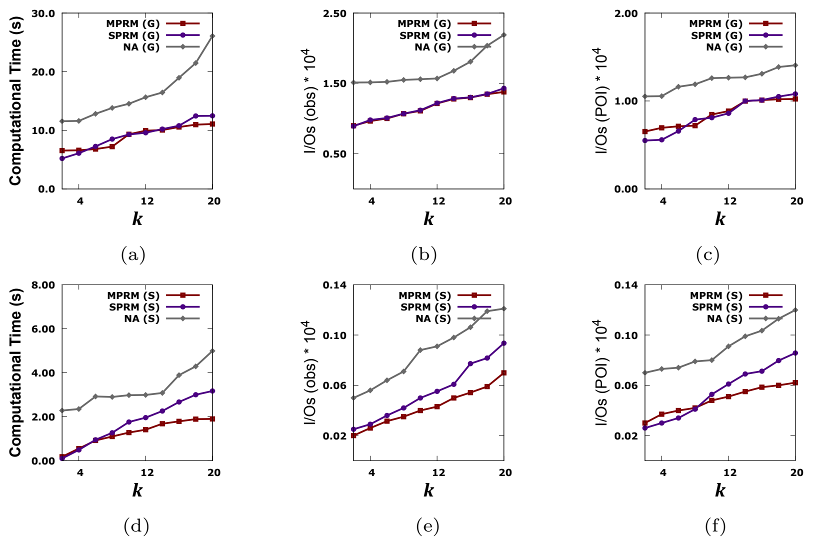


Figure 6 Effect of the number of required POIs k.

We use a 2.4 GHz Intel i5 CPU and 16 GB main memory. Table 2 shows the range and default values of our experiment parameters. To observe the effect of a parameter in an experiment, we set other parameters to their default values.

For every experiment, we consider 200 sample COkD queries and takes the average performance in terms of the computational time and I/O costs for retrieving POIs and obstacles from the database. For every COkD query sample, we randomly generate l_f and d according to the specified range in the experiment. Then we randomly generate l_cs in the following way: a user moves towards the direction of d but the followed path may not be the shortest one for arriving at d. Though the distance between two l_cs is kept fixed, the number of l_cs may vary for two paths having l_f and d in the same query range (e.g., 3000 units). Therefore, we show the average computational time and I/Os required per l_c for a path as the cost of a COkD query sample.

6.1 Effect of the Number of Required POIs k

Figure 6 shows that the required computational time and I/Os are higher for the naive approach than those for our safe region based approach for varying k. From Figures 6(a) and 6(d), we observe that the computational time increases rapidly for the naive approach than our safe region based approach for higher values of k. This is because with the increase of k, for both SPRM and MPRM, the safe regions become larger and the probability for l_c to remain inside OISR increases, which avoids the re-computation of COkD answer. On the other hand, the naive approach requires to evaluate the obstructed nearest detour POIs for every update of l_c and the time required for the evaluation increases for the higher values of k.

Figures 6(b), 6(e), 6(c) and 6(f) show that the I/O cost for both POIs and obstacles increases with the increase of k, which is expected because the number of POIs and obstacles retrieved from the database increase with the increase of k.

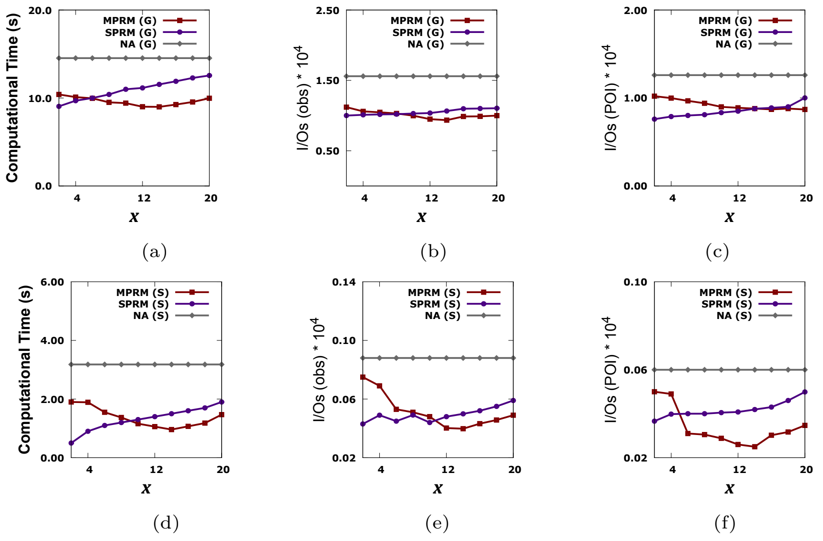


Figure 7 Effect of the number of auxiliary POIs x.

6.2 Effect of the Number of Auxiliary POIs x

Figure 7 shows that the computational time and I/O cost for the naive approach is higher than our approach but remain same irrespective of values of x because the naive approach does not retrieve auxiliary POIs. On the other hand, we observe that for MPRM the performance improves with the increase of x upto a certain threshold then again degrades. The reason is as follows. With the increase of x, the area of safe region becomes larger and the query processing overhead decreases, but after certain threshold with the increase of x, the cost for computing the non visible regions diminishes the gain achieved from the large safe regions. For SPRM, we observe that the performance degrades with the increase of x. This is because for SPRM, POIs are always retrieved with respect to l_f, and the retrieval of POIs that are not required increases with the increase of x.

6.3 Effect of the Query Range R

In this experiment, we vary R from 500 meter to 3000 meter by considering the typical travelling distance of a pedestrian. Figure 8 shows that SPRM performs better than MPRM, which can be explained from the underlying structure of SPRM and MPRM. It is expected that set of nearest detour POIs remain same for several timestamps, and the number of POIs and obstacles retrieved with respect to l_c and d is small. On the other hand, MPRM needs to compute obstructed detour distances for every element in Q_p with respect to l_c. Therefore, SPRM performs better than MPRM.

The performance of both naive and safe region based approaches degrades with the increase of R. Since the distance between consecutive l_cs increases with the increase of R, the probability that l_c falls outside the safe region also increases and more POIs and obstacles need to be retrieved from the database.

6.4 Effect of POI-Obstacle Ratio $|P|/|O|$

Figure 9 shows the comparative performance between the naive approach and the safe region based approach for varying the ratio of the number of POIs and the number of obstacles $|P|/|O|$. Increase of $|P|/|O|$ ratio means that the sample space contains more POIs than

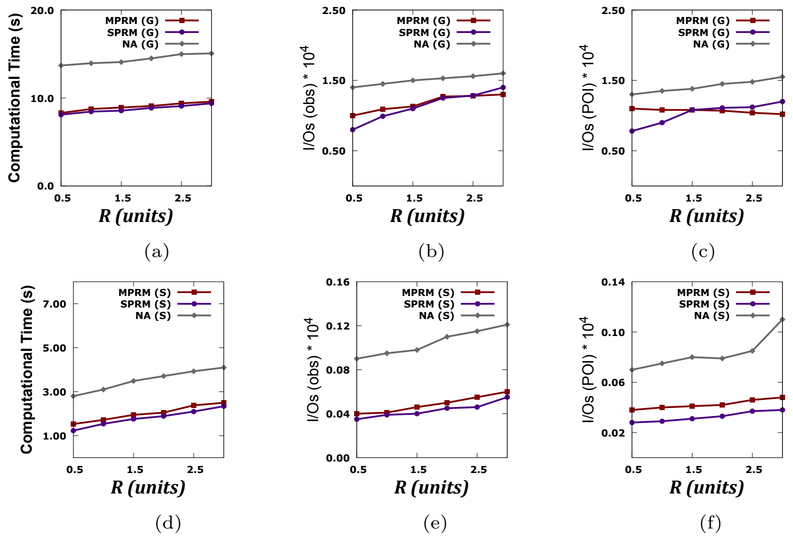


Figure 8 Effect of the query range R.

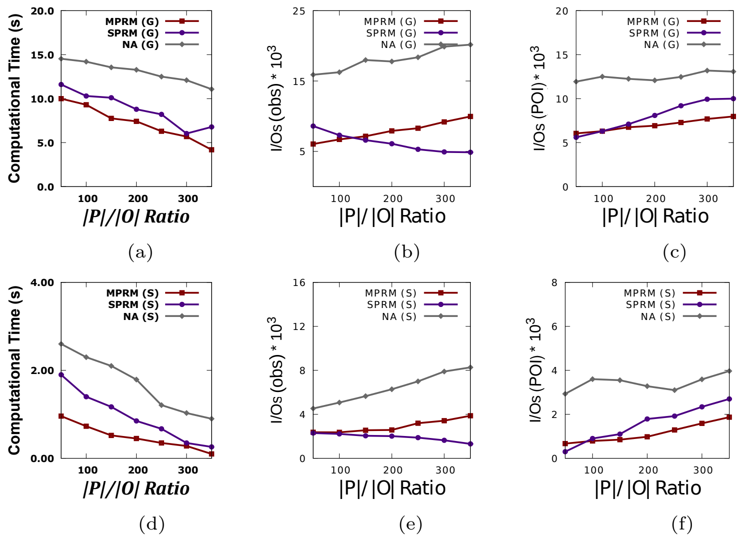


Figure 9 Effect of POI-obstacle ratio $[P]/[O]$.

obstacles. With the increase of $|P|/|O|$, the I/O cost for POIs increases for both SPRM and MPRM, which is expected. For SPRM, the I/O cost of obstacles decreases because less number of obstacles are retrieved with respect to fixed locations l_f and d. However, for MPRM, the I/O cost of obstacles increases because detour obstructed distances of POIs are computed with respect to different locations.

7 Conclusion

We have introduced and formulated COkD queries. We have proposed the first approach based on safe regions for efficient processing of COkD queries. We have further improved the efficiency of our approach by developing two POI retrieval algorithms: SPRM and MPRM.

We have performed experiments using both real and synthetic datasets. The results show that our approach for COkD queries with SPRM requires on average 67.3% less processing time, 62% less I/Os for obstacles and 72.6% less I/Os for POIs than the naive approach that applies the existing OkD query processing algorithm to evaluate for COkD queries. On the other hand, our approach with MPRM requires on average 69.2% less processing time, 67% less I/Os for obstacles and 72% less I/Os for POIs than the naive approach.

References

1 Anika Anwar and Tanzima Hashem. Optimal obstructed sequenced route queries in spatial databases. In *EDBT*, pages 522–525, 2017.

2 Takao Asano, Tetsuo Asano, Leonidas J. Guibas, John Hershberger, and Hiroshi Imai. Visibility of disjoint polygons. *Algorithmica*, 1(1):49–63, 1986.

3 Ugur Demiryurek, Farnoush Banaei-Kashani, and Cyrus Shahabi. Efficient continuous nearest neighbor query in spatial networks using euclidean restriction. In *SSTD*, pages 25–43, 2009.

4 Yunjun Gao, Jiacheng Yang, Gang Chen, Baihua Zheng, and Chun Chen. On efficient obstructed reverse nearest neighbor query processing. In *SIGSPATIAL GIS*, pages 191–200, 2011.

5 Yunjun Gao and Baihua Zheng. Continuous obstructed nearest neighbor queries in spatial databases. In *SIGMOD*, pages 577–590, 2009.

6 Yu Gu, Ge Yu, and Xiaonan Yu. An efficient method for k nearest neighbor searching in obstructed spatial databases. *J. Inf. Sci. Eng.*, pages 1569–1583, 2014.

7 Antonin Guttman. R-trees: a dynamic index structure for spatial searching. In *SIGMOD*, pages 47–57, 1984.

8 Tanzima Hashem, Lars Kulik, and Rui Zhang. Countering overlapping rectangle privacy attack for moving knn queries. *Inf. Syst.*, 38(3):430–453, 2013.

9 Paul J. Heffernan and Joseph S. B. Mitchell. An optimal algorithm for computing visibility in the plane. *SIAM J. Comput.*, 24(1):184–201, 1995.

10 Chuanwen Li, Yu Gu, Jianzhong Qi, Rui Zhang, and Ge Yu. A safe region based approach to moving knn queries in obstructed space. *KAIS*, 45:417–451, 2015.

11 Kyriakos Mouratidis, Man Lung Yiu, Dimitris Papadias, and Nikos Mamoulis. Continuous nearest neighbor monitoring in road networks. In *VLDB*, pages 43–54, 2006.

12 Sarana Nutanong, Egemen Tanin, Jie Shao, Rui Zhang, and Ramamohanarao Kotagiri. Continuous detour queries in spatial networks. *IEEE TKDE*, 24:1201–1215, 2012.

13 Sarana Nutanong, Rui Zhang, Egemen Tanin, and Lars Kulik. The v*-diagram: a query-dependent approach to moving KNN queries. *PVLDB*, 1(1):1095–1106, 2008.

14 Shuo Shang, Ke Deng, and Kexin Xie. Best point detour query in road networks. In *SIGSPATIAL GIS*, pages 71–80, 2010.

15 Nusrat Sultana, Tanzima Hashem, and Lars Kulik. Group nearest neighbor queries in the presence of obstacles. In *SIGSPATIAL GIS*, pages 481–484, 2014.

16 Nusrat Sultana, Tanzima Hashem, and Lars Kulik. Group meetup in the presence of obstacles. *Inf. Syst.*, 61:24–39, 2016.

17 Chenyi Xia, David Hsu, and Anthony KH Tung. A fast filter for obstructed nearest neighbor queries. In *BICOD*, pages 203–215, 2004.

18 Jin Soung Yoo and Shashi Shekhar. In-route nearest neighbor queries. *GeoInformatica*, 9(2):117–137, 2005.

19 Jun Zhang, Dimitris Papadias, Kyriakos Mouratidis, and Manli Zhu. Spatial queries in the presence of obstacles. In *EDBT*, pages 366–384, 2004.

20 Huaijie Zhu, Xiaochun Yang, Bin Wang, and Wang-Chien Lee. Range-based obstructed nearest neighbor queries. In *SIGMOD*, pages 2053–2068, 2016.