Poster Paper

Optimal Obstructed Sequenced Route Queries in Spatial Databases

Anika Anwar[1], Tanzima Hashem[2]
[1,2]Department of CSE, Bangladesh University of Engineering and Technology, Bangladesh
{[1]anika.anwar@yahoo.com } {[2]tanzimahasem@cse.buet.ac.bd}

ABSTRACT

We introduce optimal obstructed sequenced route (OOSR) queries, a novel query type in spatial databases. For a given source and destination locations and the sequence of required types of points of interests (POIs) (e.g., first an ATM booth then a restaurant), an OOSR query returns the locations of POIs, one from every required type, that together minimize the obstructed trip distance (OTD) from the source to the destination via the POIs. A pedestrian's walking path is obstructed by the presence of obstacles like a river, a fence or a private property, and an obstructed distance is measured as the length of the shortest path between two locations by avoiding the obstacles. We develop the first solution to address OOSR queries. We exploit elliptical properties and develop a novel OTD computation technique that does not retrieve the same obstacles multiple times, reuses the already computed obstructed distances, and minimizes the retrieval of the extra obstacles. We propose efficient algorithms to evaluate OOSR queries with reduced IO and query processing overhead. We perform experiments using a real dataset and show a comparative analysis between OOSR algorithms.

1. INTRODUCTION

The widespread usage of location aware mobile devices has expedited the proliferation of location-based services in recent years. Researchers have proposed variant of location-based queries [1, 4, 5, 6] to assist users in planning trips in an optimized manner. In this paper, we introduce a new variant of trip planning query, an optimal obstructed sequenced route (OOSR) query that allows pedestrians to plan trips with the minimum travel distance in presence of obstacles like a river, a fence or a private property in the space. For example, a tourist walking from an attraction to the hotel may want to withdraw money from an ATM booth and then have dinner at a restaurant, or a pedestrian in the city may want to buy a medicine from a pharmacy and then visit a shopping mall before going to the bus station. An OOSR query returns the location of a point of interest (POIs) for every required type (e.g., an ATM booth or a restaurant) that together minimize the obstructed trip distance (OTD) from a user's source to the destination via the POIs. We propose the first solution for OOSR queries.

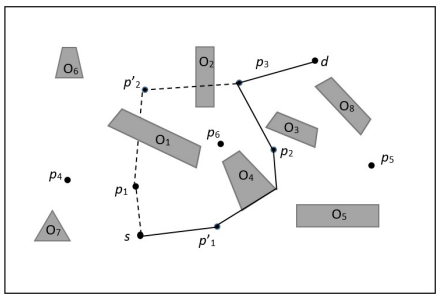


Figure 1: An example of an OOSR query

Optimal sequenced route (OSR) queries have been addressed in the unobstructed space that do not consider the presence of obstacles and cannot facilitate trip planning for pedestrians. Figure 1 shows that POIs p_1, p_2', and p_3 minimize the trip distance if obstacles are not considered. On the other hand, the answer changes for an OOSR query as in reality pedestrians cannot cross the interior of obstacles and POIs p_1', p_2, and p_3 minimize the OTD.

The efficiency of an OOSR algorithm depends on the OTD computation technique and the number of POIs explored for finding the optimal answer. The smaller the number of POIs retrieved from the database while searching for the optimal query answer, the more efficient the algorithm is. More importantly, the smaller number of POIs reduces the number of OTD computations. In summary, the contributions of our paper are summarized as follows:

- We introduce and formulate OOSR queries. To the best of our knowledge, we first address the OOSR query.
- We develop a novel OTD computation technique that (i) does not retrieve the same obstacles multiple times, (ii) reuses the already computed obstructed distances, and (iii) minimizes the retrieval of the extra obstacles.
- We combine the Euclidean lower bound and elliptical properties to prune POIs that cannot be part of the optimal answer, and develop efficient algorithms for processing OOSR queries with reduced IO and processing overheads.
- We compare the efficiency of our algorithms through extensive experiments using real datasets.

2. PROBLEM FORMULATION

An OOSR query is formally defined as follows:

Definition: Optimal Obstructed Sequenced Route (OOSR) Queries. Given a set of POIs P and a set of obstacles O in a 2-dimensional space, a source location s, a destination location d, and a set of m sequenced POI types $T = \{t_1, t_2, \ldots, t_m\}$, an OOSR query returns $A = \{p_{t_1}, p_{t_2}, \ldots, p_{t_m}\}$, a POI from every required type, where A minimizes the obstructed trip distance (OTD).

An obstacle o_i is a polygon in a 2-dimensional space and an obstructed space does not allow a pedestrian to cross the obsta-

cle like a river, a fence or a private property. An obstructed distance $dist_o(.,.)$ between two locations is measured as the length of the shortest path between two locations by avoiding the obstacles. An OTD $Tdist_o(s,d,A)$ is measured as $dist_o(s,p_{t_1}) + \sum_{i=1}^{m-1} dist_o(p_{t_i},p_{t_{i+1}}) + dist_o(p_{t_m},d)$.

The set of POIs P and the set of obstacles O are indexed using two separate R-trees, POI RTree and Obstacle RTree, respectively, in the database of a location-based service provider (LSP). When a user requests an OOSR query to the LSP, the LSP evaluates the OOSR query and returns the answer to the user.

3. RELATED WORK

Trip planning queries [4] and variants [5, 6] have been extensively studied in the literature. A trip planning algorithm was introduced in [4], where a user can visit POIs in any sequence that minimizes the trip distance. In [6], the authors first addressed an optimal sequenced route (OSR) query that allows users to specify the sequence of visiting POI types. In [7], the authors focus on protecting location privacy of users while evaluating an optimal trips. However, none of the above approaches consider the presence of obstacles while evaluating the queries.

Researchers have recently focused on developing algorithms for processing variant queries in the obstructed space. In [3, 10], the authors proposed algorithms for processing nearest neighbor queries in the obstructed space. The approaches in [8, 9] evaluate group nearest neighbor queries in the presence of obstacles, whereas the focus of [2] is on obstructed reverse nearest neighbor queries.

Algorithm 1 CompOTD($s,d,pTrip$)

Input: s, d, and a set of POIs $pTrip = \{p_{t_1}, p_{t_2}, \ldots, p_{t_m}\}$
Output: The obstructed trip distance $Tdist_O(s,d,pTrip)$

1: **if** $dist_E(s,p_{t_1}) > dist_E(d,p_{t_m})$ **then**
2: $d_{max} \leftarrow dist_E(s,p_{t_1})$
3: **else**
4: $d_{max} \leftarrow dist_E(d,p_{t_m})$
5: **end if**
6: **for** $i \leftarrow 1$ to $m-1$ **do**
7: $j \leftarrow i+1$
8: $d_{ij} \leftarrow ComputeMin(s,d,p_{t_i},p_{t_j})$
9: **if** $d_{ij} + dist_E(p_{t_i},p_{t_j}) > d_{max}$ **then**
10: $d_{max} \leftarrow d_{ij} + dist_E(p_{t_i},p_{t_j})$
11: **end if**
12: **end for**
13: **repeat**
14: $d_{prev} \leftarrow d_{max}$
15: $a \leftarrow 2 \times \frac{d_{max}}{(1-e)}$
16: $O \leftarrow IOR(s,d,a)$
17: $VG \leftarrow ConstructVG(s,d,p_{t_1},p_{t_2}\ldots,p_{t_m},O)$
18: $dist_o(s,p_{t_1}) \leftarrow CompObsDist(VG,s,p_{t_1})$
19: $dist_{sum} \leftarrow 0$
20: **for** $i \leftarrow 1$ to $m-1$ **do**
21: $j \leftarrow i+1$
22: $dist_o(p_{t_i},p_{t_j}) \leftarrow CompObsDist(VG,p_{t_i},p_{t_j})$
23: $dist_{sum} \leftarrow dist_{sum} + dist_o(p_{t_i},p_{t_j})$
24: **end for**
25: $dist_o(d,p_{t_m}) \leftarrow CompObsDist(VG,d,p_{t_m})$
26: $Tdist_O(s,d,t) \leftarrow dist_o(s,p_{t_1}) + dist_{sum} + dist_o(d,p_{t_m})$
27: $d_{max} \leftarrow Tdist_O(s,d,pTrip)$
28: **until** $d_{max} == d_{prev}$
29: **return** $Tdist_O(s,d,pTrip)$

4. AN OTD COMPUTATION TECHNIQUE

A major challenge of a query processing algorithm in the obstructed space is the complexity of computing the obstructed distance. The obstructed distance is computed as the length of the shortest path between two locations by avoiding the obstacles. There exist algorithms [10] to compute the obstructed distance between two locations. However, computing obstructed distances for pairs of locations independently by applying an existing algorithm requires performing the same computations and the retrieval of same obstacles from the database multiple times. To overcome the limitations, different optimization techniques [2, 9] have been developed in the context of obstructed group nearest neighbor (OGNN) and obstructed reverse nearest neighbor (ORNN) queries, which are not applicable for OOSR queries.

Evaluating an OOSR query requires the computation of a large number of OTDs, and an OTD is the summation of a number of obstructed distances. We develop a novel OTD computation technique that incrementally expands the obstacle retrieval area as an elliptical shape. We develop a technique to compute the length of the major axis of the ellipse to guarantee that obstacles required for every obstructed distance computation are simultaneously retrieved. Furthermore, we reuse the already retrieved obstacles and computed obstructed distances for computing a new OTD. The intuition behind using an elliptical region instead of any other shape is to increase the probability of reusing the already retrieved obstacles, and minimizing the retrieval of obstacles that are not required for obstructed distance computations. We will show in the next section that the refined POI search space in our proposed OOSR algorithms expands as an elliptical region, and therefore there is a high probability that the retrieved POI falls inside the area of the already retrieved obstacles and the obstructed distances involving the POI can be computed using already retrieved obstacles.

We use the existing technique [10] to compute the obstructed distance between two points using a visibility graph. The vertices of a visibility graph are the corner points of polygons representing the obstacles and the locations between which the obstructed distance needs to be computed. There is an edge between two vertices if no obstacle crosses the direct path between those vertices. The obstructed distance between two locations is the length of the shortest path between two vertices representing the locations. It is not feasible to pre-compute a visibility graph for a large set of obstacles. We only retrieve those obstacles from the database that are relevant to the OOSR query and construct the visibility graph.

Algorithm 1 shows the pseudocode for computing an OTD. Without loss of generality, we explain the steps of computing $Tdist_o(s,d,p_1,p_2)$ for an example shown in Figure 2. The algorithm computes $dist_o(s,p_1)$, $dist_o(p_1,p_2)$, and $dist_o(p_2,d)$ simultaneously. Using the function *ComputeMin* in Line 8, the algorithm finds the Euclidean distance $dist_E(p_2,s)$ as the minimum among $dist_E(p_1,s)$, $dist_E(p_2,s)$, $dist_E(p_1,d)$, and $dist_E(p_2,d)$. Thus, $dist_E(p_2,s)$ is assigned to d_{12} and p_2 becomes the center of the circle used for computing $dist_o(p_1,p_2)$ as shown in Figure 2(a).

In the next step, to compute $dist_o(s,p_1)$, $dist_o(p_1,p_2)$, and $dist_o(p_2,d)$, the algorithm retrieves obstacles inside the circles centered at s, p_2 and d with radius $dist_E(s,p_1)$, $dist_E(p_1,p_2)$, and $dist_E(p_2,d)$, respectively. Figure 2(a) shows that there are overlaps among the circles. Thus, to avoid the retrieval of same POIs multiple times, our algorithm computes an ellipse with foci at s and d that includes three circles, and retrieves obstacles in the ellipse as shown in Figure 2(b). To ensure the inclusion of the circles, the periapsis, i.e., the smallest radial distance of the ellipse needs to be greater than or equal to d_{max}, where d_{max} is the maximum of $dist_o(s,p_1)$, $d_{12} + dist_o(p_1,p_2)$, and $dist_o(p_2,d)$. Thus, the length

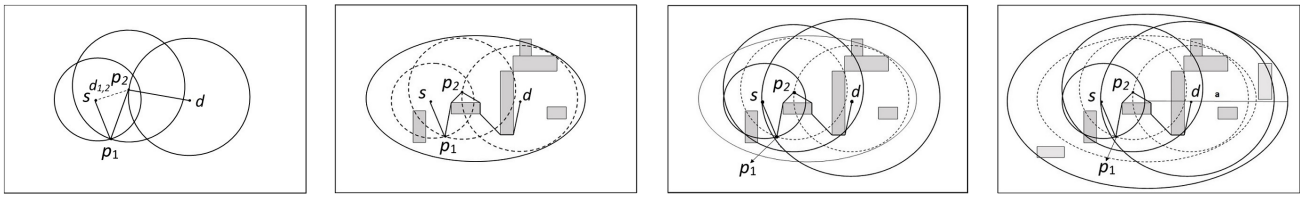


Figure 2: Steps of computing $Tdist_o(s,d,p_1,p_2)$

of the major axis is computed as $2 \times \frac{d_{max}}{(1-e)}$, where eccentricity e is determined in experiments. The function *IOR* in Line 16 incrementally retrieves nearest obstacles with respect to s and d, where the distance is measured as the summation of the minimum Euclidean distances of the obstacle from s and d, respectively.

The algorithm constructs the visibility graph and computes $dist_o(s,p_1)$, $dist_o(p_1,p_2)$, and $dist_o(p_2,d)$ based on the retrieved obstacles. In Figure 2(c), we see that the radius of the circles centered at p_2 and d increases to $dist_o(p_1,p_2)$ and $dist_o(p_2,d)$ from $dist_E(p_1,p_2)$ and $dist_E(p_2,d)$, respectively. Since $dist_o(s,p_1)$ and $dist_E(s,p_1)$ are equal, the circle centered at s does not change and $dist_o(s,p_1)$ is finalized. The algorithm again retrieves obstacles so that the new ellipse includes the circles as shown in Figure 2(d). We observe that in Figure 2(d), though new obstacles are retrieves but those obstacles do not increase any of the obstructed distance. Thus, $dist_o(p_1,p_2)$, and $dist_o(p_2,d)$ are finalized and $Tdist_o(s,d,p_1,p_2)$ is computed in Line 26.

Algorithm 2 RRB_OOSR(s,d,T)

Input: A source s, a destination d, required POI types T
Output: The answer set A

```
1: A_initial ← RetrieveInitialPOIs(s,d,T)
2: POITrips ← CompTrips(A_initial)
3: POITrips_prev ← POITrips
4: MinTDist_o ← ∞
5: for each pTrip ∈ POITrip do
6:     TDist_o ← CompOTD(s,d,pt)
7:     if TDist_o < MinTDist_o then
8:         MinTDist_o ← TDist_o
9:         A ← pTrip
10:    end if
11: end for
12: Maxd ← FindMaxDist(A_initial)
13: if Maxd < MinTDist_o then
14:    A_range ← RetrievePOIs(s,d,T,MinTDist_o)
15:    POITrips ← CompNewTrips(A_initial,A_range,POITrips_prev)
16:    for each pTrip ∈ POITrips do
17:        TDist_o ← CompOTD(s,d,pTrip)
18:        if TDist_o < MinTDist_o then
19:            MinTDist_o ← TDist_o
20:            A ← pTrip
21:        end if
22:    end for
23: end if
24: return A
```

5. OOSR ALGORITHMS

In this section, we present efficient algorithms for processing OOSR queries. We develop a pruning technique to refine the POI search space by exploiting the Euclidean lower bound and elliptical properties. A POI outside the refined POI search space cannot provide the minimum OTD. The number of possible trips and OTD computations decrease with the smaller number of retrieved POIs from the database, i.e., the smaller POI search space.

According to the Euclidean lower bound property, the Euclidean trip distance is smaller or equal to the OTD. On the other hand, according to the elliptical property, the Euclidean distance between two foci of an ellipse via a POI outside the ellipse is greater than or equal to the length of the major axis of the ellipse. In our OOSR algorithms, we represent the POI search space using an ellipse, where the foci of the ellipse are at the source and destination locations of a user, and the length of the major axis of the ellipse is equal to the upper bound of the OTD. Thus, POIs outside the ellipse cannot further minimize the OTD.

We propose two OOSR algorithms: *RRB_OOSR* (range retrieval based OOSR) and *IRB_OOSR* (incremental retrieval based OOSR). The key difference between our algorithms, *RRB_OOSR* and *IRB_OOSR*, is that *RRB_OOSR* computes the upper bound of the OTD, refines the POI search space once, and then retrieves all POIs inside the POI search region using a range query. On the other hand, *IRB_OOSR* incrementally retrieves POIs and gradually refines the search space. The advantage of *IRB_OOSR* is that it retrieves less number of POIs than *RRB_OOSR*.

Both *RRB_OOSR* and *IRB_OOSR* use a heuristic [7] to compute the upper bound of the OTD. The heuristic retrieves an initial set of POIs $A_{initial}$ that includes the nearest POI of every required type from s and d. The Euclidean aggregate distance (EAD) of a POI from s and d is computed as the summation of Euclidean distances of the POI from s and d, respectively. In addition to the nearest POI of every required POI type, $A_{initial}$ also includes other POIs of required types that have EAD smaller than or equal to the maximum of EADs of the nearest POIs from every required type.

Algorithm 2 shows the pseudocode for *RRB_OOSR*. The algorithm retrieves initial POIs as $A_{initial}$ using the heuristic (Line 1), computes the sets of possible combinations of POIs as *POITrips* (Line 2), determines the OTD with respect to s and d for every set using Algorithm 1 (Line 6), and finds the upper bound of the OTD as $MinTDist_o$ in Line 8.

Next the algorithm computes *Maxd* in Line 12. The POIs that falls inside the ellipse with foci s and d, and the major axis equal to *Maxd* have been already retrieved. If $Maxd \geq MinTDist_o$, the trip with the minimum OTD has been already found because a POI that has EAD greater or equal to *Maxd* cannot further minimize $MinTDist_o$. Otherwise, the algorithm retrieves all POIs in the refined POI search space (i.e., the POIs whose EADs from s and d are smaller than $MinTDist_o$), computes the set of new combination of POIs by excluding the combinations that have already considered ($POITrips_{prev}$), and finds the set of POIs that provide the minimum OTD with respect to s and d.

Algorithm 3 shows the pseudocode for *IRB_OOSR*. Instead of retrieving all POIs in the refined POI search space, the algorithm incrementally retrieves the next nearest POI with the smallest EAD *Maxd* from s and d (Line 14). After retrieving a new POI, the algorithm further minimizes the upper bound of the OTD as $MinTDist_o$ in Line 20, if possible. The incremental retrieval of POIs continues until the condition $Maxd < MinTDist_o$ is satisfied.

Algorithm 3 IRB_OOSR(s,d,T)

Input: A source s, a destination d, required POI types T
Output: The answer set A

```
1: A_init ← RetrieveInitialPOIs(s,d,T)
2: POITrips ← CompTrips(A_initial)
3: POITrips_prev ← POITrips
4: MinTDist_o ← ∞
5: for each pTrip ∈ POITrips do
6:     TDist_o ← CompOTD(s,d,pTrip)
7:     if TDist_o < MinTDist_o then
8:         MinTDist_o ← TDist_o
9:         A ← pTrip
10:    end if
11: end for
12: Maxd ← FindMaxDist(A_initial)
13: while Maxd < MinTDist_o do
14:    p ← RetrieveNextPOI(s,d,T)
15:    Maxd ← dist(s,p) + dist(p,d)
16:    POITrips ← CompNewTrips(A_initial, p, POITrips_prev)
17:    for each pTrip ∈ POITrips do
18:        TDist_o ← CompOTD(s,d,pTrip)
19:        if TDist_o < MinTDist_o then
20:            MinTDist_o ← TDist_o
21:            A ← pTrip
22:        end if
23:    end for
24: end while
25: return A
```

Table 1: Experimental Setup

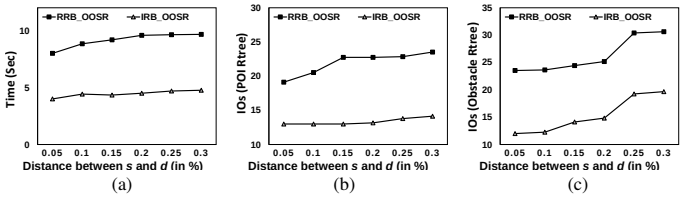
Parameter	Range	Default Value
Distance between s and d	0.05% to 0.3%	0.15%
Total POI types	10, 15, 20, 25, 30	20
Required POI types	1, 2, 3, 4, 5	3

6. EXPERIMENTS

Since we first address OOSR queries, we compare our proposed algorithms through experiments. We vary the distance between s and d, the number of total POI types in the POI data set and the number of required POI types in the query. Table 2 shows the range and default value of each parameter that we used in our experiments. When we vary a parameter in an experiment, we set other parameters to their default values. We used the real dataset of Germany, which consists of 34334 minimum bounded rectangles (MBRs) of railway lines (rrlines) that represent obstacles and 307992 MBRs of hypsography data (hypsogr) that represent POIs in our experiments. We normalized the total space into 10,000 $\times$ 10,000 square units. We conducted each experiment for 50 samples of OOSR queries and obtained the average experimental results. We measured the processing time and IO cost using an Intel(R) Core i5-5200U CPU (2.20 GHz) with 4 GB RAM.

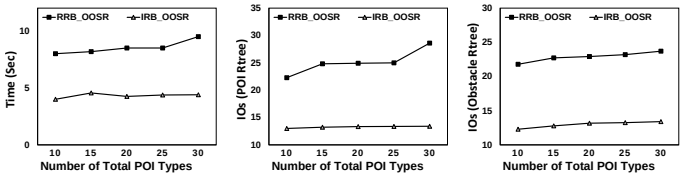
Initially, we varied the values of eccentricity of the ellipse e as 0, 0.25, 0.5, 0.75 and 1, and run experiments for the default values of other parameters. We found that the algorithms perform better in terms of time and IOs for the value of $e = 0.75$. Therefore we set this value as the default eccentricity (e) in our experiments.

Figure 3 shows that the processing time and IOs increases for both of our algorithms with the increase of the distance between s and d. This is because when the distance between s and d increases, the areas for retrieving POIs and obstacles also increase. We also observed that *IRB_OOSR* performs better in terms of both time and IO cost than *RRB_OOSR*. This is expected as *RRB_OOSR* retrieves more POIs than *IRB_OOSR*. Figures 4 and 5 show the similar trends for varying the number of total POI types and the number of required POI types, respectively.



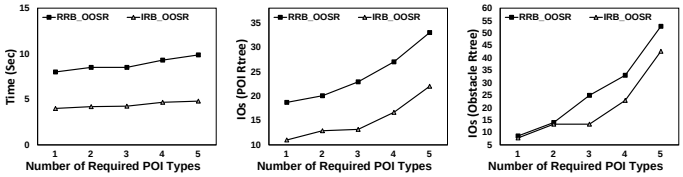
(a) (b) (c)

Figure 3: Effect of the distance between s and d in %



(a) (b) (c)

Figure 4: Effect of the number of total POI types



(a) (b) (c)

Figure 5: Effect of the number of required POI types

7. CONCLUSION

We developed a novel OTD computation technique, and OOSR algorithms: *RRB_OOSR* and *IRB_OOSR*. Experiments show that our approach can evaluate OOSR queries in real time, and on average *IRB_OOSR* requirers 2.1 times less processing time and 1.7 times less IOs than *RRB_OOSR* to process OOSR queries.

Acknowledgments

This research is supported by Bangladesh University of Engineering and Technology (BUET) and University of Asia Pacific (UAP).

8. REFERENCES

[1] H. Chen, W. Ku, M. Sun, and R. Zimmermann. The multi-rule partial sequenced route query. In *SIGSpatial*, pages 10:1–10:10, 2008.

[2] Y. Gao, J. Yang, G. Chen, B. Zheng, and C. Chen. On efficient obstructed reverse nearest neighbor query processing. In *GIS*, pages 191–200, 2011.

[3] Y. Gao, B. Zheng, G. Chen, C. Chen, and Q. Li. Continuous nearest-neighbor search in the presence of obstacles. *ACM Trans. Database Syst.*, 36(2):9:1–9:43, 2011.

[4] F. Li, D. Cheng, M. Hadjieleftheriou, G. Kollios, and S. Teng. On trip planning queries in spatial databases. In *SSTD*, pages 273–290, 2005.

[5] Y. Ohsawa, H. Htoo, N. Sonehara, and M. Sakauchi. Sequenced route query in road network distance based on incremental euclidean restriction. In *DEXA*, pages 484–491, 2012.

[6] M. Sharifzadeh, M. Kolahdouzan, and C. Shahabi. The optimal sequenced route query. *The VLDB Journal*, 17(4):765–787, 2008.

[7] S. C. Soma, T. Hashem, M. A. Cheema, and S. Samrose. Trip planning queries with location privacy in spatial databases. *World Wide Web*, 2016.

[8] N. Sultana, T. Hashem, and L. Kulik. Group nearest neighbor queries in the presence of obstacles. In *SIGSPATIAL*, pages 481–484, 2014.

[9] N. Sultana, T. Hashem, and L. Kulik. Group meetup in the presence of obstacles. *Inf. Syst.*, 61:24–39, 2016.

[10] J. Zhang, D. Papadias, K. Mouratidis, and M. Zhu. Spatial queries in the presence of obstacles. In *EDBT*, pages 366–384, 2004.