Group Nearest Neighbor Queries in the Presence of Obstacles

Nusrat Sultana
Department of CSE
Bangladesh university of Engineering and Technology
Dhaka, Bangladesh
nusratst.cse@gmail.com

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

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

ABSTRACT

In this paper, we introduce obstructed group nearest neighbor (OGNN) queries, that enable a group to meet at a point of interest (e.g., a restaurant) with the minimum aggregate travel distance in an obstructed space. In recent years, researchers have focused on developing algorithms for processing GNN queries in the Euclidean space and road networks, which ignore the impact of obstacles such as buildings and lakes in computing distances. We propose the first comprehensive approach to process an OGNN query. We present an efficient algorithm to compute aggregate obstructed distances, which is an essential component for processing OGNN queries. We exploit geometric properties to develop pruning techniques that reduce the search space and incur less processing overhead. We validate the efficacy and efficiency of our solution through extensive experiments using both real and synthetic datasets.

Categories and Subject Descriptors

H.2.8 [**Database Applications**]: Spatial databases and GIS; H.2.4 [**Systems**]: Query processing

General Terms

Algorithms, Design, Experimentation

Keywords

Group nearest neighbor queries, Obstacle, Visibility graph

1. INTRODUCTION

An important class of information and enquiry service is a group nearest neighbor (GNN) query [8, 9] that enables a group to meet with the minimum aggregate travel distance. Through a GNN query a group of friends, located at different places in a city center, can determine the location of a common point of interest (POI) such as a restaurant, a shopping mall or a movie theater that minimizes their *total travel distance* or the *maximum travel distance* for all group members. Minimizing the maximum travel distance enables the group to meet at a POI within the *shortest possible time*.

SIGSPATIAL'14, November 04 - 07 2014, Dallas/Fort Worth, TX, USA
Copyright 2014 ACM 978-1-4503-3131-9/14/11 ...$15.00
http://dx.doi.org/10.1145/2666310.2666484

GNN queries have been addressed in the Euclidean space [4, 8] and road networks [9, 11], which did not consider the impact of obstacles on the travel distance. For example, a park has a number of obstacles such as trees, lakes or fences that obstruct a pedestrian's direct path and require a detour. Similarly, in a city center some buildings might restrict a pedestrian's direct path or some roads might only permit vehicles. Current GNN queries are tailored to Euclidean space or road networks but ignore the impact of obstacles. In this paper, we introduce a novel form of GNN queries in the presence of obstacles, which we call *Obstructed Group Nearest Neighbor (OGNN)* queries. To the best of our knowledge, we propose the first comprehensive approach to find group nearest neighbors in the presence of obstacles.

Although there exists obstructed nearest neighbor (ONN) algorithms [12] in the obstructed space, a straightforward application of an ONN algorithm to evaluate a OGNN query equals to an exhaustive search and makes computations prohibitively expensive. Such a straightforward technique requires to incrementally evaluate obstructed nearest neighbors (i.e., POIs) for every query point (i.e., a group member's location) independently and compute the obstructed aggregate distance for each retrieved POI until the actual OGNN has been identified.

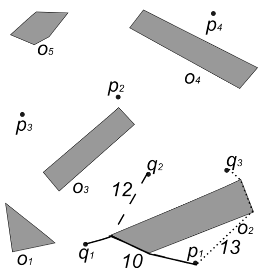
The smaller the number of POIs that an OGNN algorithm has to consider for computing the actual OGNN, the more efficient the algorithm is because for every retrieved POI, an OGNN algorithm has to compute the aggregate obstructed distance with respect to the query points. To develop an efficient approach for OGNN queries, we focus on developing (i) pruning techniques to refine the search space and (ii) efficient algorithms to compute aggregate distances under the presence of obstacles.

We summarize our key contributions as follows:

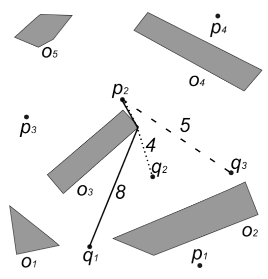
- We introduce GNN queries in the obstructed space.
- We develop different pruning techniques for OGNN queries and based on these pruning techniques, we propose two algorithms, a *GBQM* (Group Based Query Method) and a *CBQM* (Centroid Based Query Method), to efficiently evaluate OGNN queries.
- We develop an efficient technique to compute aggregate obstructed distances for POIs with respect to the locations of the group members.
- We conduct an extensive experimental analysis using both real and synthetic datasets and show a comparative analysis between our algorithms based on different parameters.

2. PROBLEM SETUP

The *obstacle path problem* [1] finds the shortest path between two points p and q in presence of obstacles, where obstacles are



(a) $f = \text{SUM}$



(b) $f = \text{MAX}$

Figure 1: An OGNN query example

represented with non-overlapping 2D polygons and the path does not cross the interior of any obstacle. The length of the obstructed path connecting two points p and q is called the *obstructed distance* and denoted as $dist_O(p,q)$.

The *aggregate obstructed distance*, $dist_{AO}(p,Q)$, between a data point[1] p and a set of query points (i.e., user locations) $Q = \{q_1, q_2, \ldots, q_n\}$ is computed with respect to an aggregate function f. The function f can be SUM that minimizes the total obstructed travel distance or MAX that minimizes the maximum obstructed travel distance. Figure 1(a) and 1(b) show examples of OGNN queries ($k = 1$) that return p_1 and p_2 as answers for aggregate functions SUM and MAX, respectively. A k obstructed group nearest neighbor query (kOGNN) is formally defined as follows:

DEFINITION 1. ***(kOGNN Queries).*** *Given a set of n query points $Q = \{q_1, q_2, \ldots, q_n\}$, a set of m data points $P = \{p_1, p_2, \ldots, p_m\}$, and a set of h obstacles $O = \{o_1, o_2, \ldots, o_h\}$, a kOGNN query returns k points from P, which have the k smallest values for an aggregate function $f : Q \mapsto \mathbb{R}$.*

3. RELATED WORK

Efficient algorithms [4, 8, 9, 11] have been developed for GNN queries in Euclidean space and road networks. In Euclidean space, the trip distance is measured ignoring the presence of obstacles. In road networks, the movement is permitted in a predefined structure and the distance is measured as the length of the shortest path between two points. In an obstructed space obstacles determine areas that cannot be crossed and the distance is measured as the length of the shortest path between two points in presence of obstacles.

Recently researchers have focused on developing algorithms for processing range, nearest neighbor, and continuous nearest neighbor queries in the obstructed space [12, 2]. However, none of these spatial query processing techniques in the obstructed space consider query from a group of users and instead focus on individuals. Group visible nearest neighbor queries [10] retrieve data points whose travel paths from a query point are not blocked by any obstacle, which is different from OGNN queries.

4. AN AGGREGATE OBSTRUCTED DISTANCE COMPUTATION TECHNIQUE

There exists algorithm [12] for computing the obstructed distance between two points p and q. To compute an aggregate obstructed distance we can use this algorithm to first compute the obstructed distance between a data point p and a query point $q \in Q$ and then compute the aggregate obstructed distance by applying the aggregate function to these individual obstructed distances. However, this straightforward technique would require to retrieve the same obstacles multiple times from the database and incur high computational overhead. To overcome this limitation, we develop *Multi Point Aggregate Obstructed Distance* (MPAOD) computation technique to compute aggregate obstructed distances.

[1]We use the term POI and data point interchangeably.

We use the *visibility graph* [6] to compute aggregate obstructed distances. A visibility graph consists of nodes representing point location of data points, obstacle vertices, and query points. There is an edge between two nodes if and only if the two nodes are mutually visible, i.e., there is no obstacle edge obstructing the visibility between them. If the line segment connecting two locations does not pass through any obstacle, then we say that the two points are visible to each other and we connect them with an edge in the visibility graph. An important property of the visibility graph is the length of the shortest path between two nodes in the visibility graph represents the obstructed distance between two locations represented by those nodes [6].

In our current problem, due to the large size of spatial datasets, it is not feasible to keep the whole visibility graph in main memory. We use the algorithm proposed in [12] to incrementally construct the visibility graph. We add only those obstacles and data points in the graph that are relevant to the query. We update the graph when new data points or obstacles arrive. We remove data points or obstacles that are not relevant to our query to keep the graph small.

4.1 MPAOD

Algorithm 1 shows the pseudocode for MPAOD. The input to the algorithm are a set of query points Q, a data point p, an obstacle RTree RT_O and a local visibility graph LVG. The output of the algorithm is $dist_{AO}(p,Q)$, the aggregate obstructed distance between the data point and the group of users.

Algorithm 1: *CompAggObsDist*(Q, p, RT_O, LVG)

```
Input: A set of query points $Q = (q_1, q_2, \ldots q_n)$, a data point
       $p$, an obstacle RTree $RT_O$, a local visibility graph $LVG$
Output: The aggregate obstructed distance $dist_{AO}(p,Q)$
1  foreach $q \in Q$ do
2      $dist_O(p,q_i) = dist_E(p,q_i)$
3  repeat
4      $d_{max} \leftarrow \max_{1 \le i \le n} dist_O(p,q_i)$
5      $O \leftarrow$ IOR $(p, RT_O, d_{max})$
6      foreach $o \in O$ do
7          foreach $q \in Q$ do
8              if $o$ intersects $SP_{p,q}$ then
9                  Add $q$ in $L_Q$
10                 Add $o$ in $LVG$
11     foreach $q \in L_Q$ do
12         $dist_O(p,q)$=$compObsDist(q,p,LVG)$
13 until $L_Q = \emptyset$
14 $dist_{AO} \leftarrow f_{i=1}^{n} dist_O(p,q_i)$
15 return $dist_{AO}$
```

The algorithm at first computes individual Euclidean distances between the data point p and each of the query points $q \in Q$ and assigns them as the initial obstructed distances between p and $q \in Q$, respectively (Lines 1-2). Next the algorithm finds the maximum obstructed distance computed from individual obstructed distances as d_{max} (Line 4). Then it incrementally retrieves all obstacles that are within the distance d_{max}, centering the data point p by using a function $IOR(p, RT_O, d_{max})$ (Line 5).

However, after retrieving the obstacles, the algorithm filters out those obstacles which do not intersects any of the already computed shortest path $SP_{p,q}$ between the data point p and a query point $q \in Q$. It also stores query points in a set L_Q, whose obstructed distances need to be recomputed (Lines 6-10). The re-computation of the obstructed distance between a data point p and a query point q is required only when the shortest path between p and q intersects any obstacles retrieved by the incremental obstacle retrieval.

After filtering out unnecessary obstacles the algorithm updates the visibility graph with the new obstacles and re computes the obstructed distances between p and all the query points $q \in L_Q$ (Lines 7-12). The procedure repeats until the shortest path intersects no new obstacles or L_Q is empty (Lines 11-13).

Finally, the algorithm applies the aggregate function f (SUM/MAX) to compute $dist_{AO}(p,Q)$ (Lines 14-15).

5. OGNN ALGORITHMS

We develop two efficient algorithms: *CBQM* (Centroid Based Query Method) and *GBQM* (Group Based Query Method) for processing *k*OGNN queries. The key difference between them is the way they incrementally retrieve data points from the database.

5.1 CBQM

CBQM incrementally retrieves Euclidean nearest neighbors (NNs) from the geometric centroid c_Q of Q until k OGNNs have been identified. With the incremental retrieval of data points (i.e., NNs), CBQM gradually refines the search region and the search terminates when all data points inside the required search region have been explored. The refinement of the search region and the terminating condition differ depending on the aggregate function.

5.1.1 CBQM_SUM

Algorithm *CBQM_*SUM first retrieves k NNs with respect to c_Q, computes the aggregate obstructed distance using MPAOD discussed in Section 4 for the retrieved k data points, initializes the answer set A with the retrieved k data points. In the next step, the algorithm computes d_{kmax} and d_{Emax}, which are defined as follows:

d_{kmax}**:** the aggregate obstructed distance of the current k^{th} best OGNN from the retrieved data points.

d_{Emax}**:** the maximum Euclidean distance between the centroid c_Q and the data points retrieved so far.

In [7], the authors have shown that a data point outside the circle centered at c_Q with radius r, have an aggregate Euclidean distance greater or equal to $n \times r$ with respect to Q, where $n = |Q|$. Thus, any data point p' outside the circle centered at c_Q with radius $\frac{d_{kmax}}{n}$ has an aggregate Euclidean distance greater than d_{kmax}. Further, the Euclidean lower bound property states that the obstructed distance is always greater or equal than the Euclidean distance. Thus, p' has an aggregate obstructed distance greater than d_{kmax}.

Therefore, the circle centered at c_Q with radius $\frac{d_{kmax}}{n}$ represents the required search area. To ensure that *CBQM_*SUM finds k OGNNs with k smallest aggregate distances, *CBQM_*SUM requires to consider all data points located in the circle centered at c_Q with radius $\frac{d_{kmax}}{n}$. On the other hand, the data points located within the circle centered at c_Q with radius d_{Emax} have been already retrieved from the database and considered in computing d_{kmax}.

After computing d_{kmax} and d_{Emax} from initially retrieved k data points, the algorithm checks whether the terminating condition $d_{Emax} \geq \frac{d_{kmax}}{n}$ is satisfied, i.e., if all data points with an aggregate obstructed distance smaller or equal to d_{kmax} have been retrieved. If the condition is satisfied, then the current data points in A represent the actual k OGNNs. Otherwise, the algorithm incrementally retrieves the next NN p with respect to c_Q, computes $dist_{AO}(p,Q)$, and updates d_{Emax}, d_{kmax}, and A. With the incremental retrieval of the next NNs, d_{Emax} increases and the radius, $\frac{d_{kmax}}{n}$ of the required circular area decreases, if d_{kmax} is updated. The process repeats until the terminating condition is satisfied.

5.1.2 CBQM for MAX

Algorithm *CBQM_*MAX incrementally retrieves NNs from c_q in a similar way as Algorithm *CBQM_*SUM. After having an initial aggregate distance d_{kmax}, the algorithm refines the search area as the intersection of circles $\{C_1, C_2, \ldots, C_n\}$, where C_j is centered at q_j and has radius equal to d_{kmax}. In [7], the authors have shown that, for an aggregate function MAX, a data point p' outside the intersection of circles $\{C_1, C_2, \ldots, C_n\}$ has aggregate Euclidean distance greater than d_{kmax}. Since the obstructed distance of a data point is always greater or equal than the Euclidean distance, the aggregate obstructed distance of p' is also greater than d_{kmax}.

Let d_{max} represent the distance between c_q and the farthest query point in Q from c_q. *CBQM_*MAX terminates if $d_{Emax} \geq (d_{max} + d_{kmax})$, i.e., the circle centered at c_q with radius d_{Emax} covers circles $\{C_1, C_2, \ldots, C_n\}$. Thus, the retrieved data point p in an iteration may or may not be in the intersection of circles $\{C_1, C_2, \ldots, C_n\}$. If $p \in \bigcap_{j=1}^{n} C_j$, the algorithm updates d_{kmax} and A.

5.2 GBQM

GBQM incrementally retrieves Euclidean group nearest neighbors (GNNs) from Q until k OGNNs have been identified. The key idea of GBQM is based on the Euclidean lower bound property, i.e., the aggregate obstructed distance of a data point is greater or equal to the aggregate Euclidean distance. GBQM can work for both aggregate functions SUM and MAX in the similar way.

The algorithm starts with retrieving the first k GNNs with respect to Q. Then the algorithm initializes A with the retrieved data points and computes their aggregate obstructed distances and the k^{th} maximum aggregate obstructed distance, d_{kmax}. If k^{th} aggregate Euclidean distance is equal to d_{kmax}, then there is no obstacle that obstructs the path from query points to the GNNs and the algorithm has identified the OGNNs. Otherwise, the algorithm retrieves more GNNs incrementally until the Euclidean aggregate distance of the last retrieved GNN becomes greater than or equal to d_{kmax}.

Parameter	Range	Default
Group size	2, 4, 8, 16, 32	8
k	2, 4, 8, 16, 32	4
Query rectangle area	.002% to .01%	.005%
Cardinalities	$0.01\|O\|$ to $10\|O\|$	$2.5\|O\|$

Table 1: Experiment Setup

6. EXPERIMENTS

We evaluate and compare the performance of our proposed algorithms through extensive experiments. We vary the group size, number of required obstructed group nearest neighbor k and the query rectangle area. The query rectangle area represents the area of the query point distribution in the total space. We also vary the cardinalities of the synthetic dataset i.e. we control the density of the data point RTree ranging from $0.01|RT_O|$ to $10|RT_O|$ with respect to the obstacle RTree RT_O. Table 1 summarizes the range and default values for each parameter used in our experiments.

We use both real and synthetic datasets in our experiments. We use the real dataset of *Germany*, which consists of 30674 minimum bounded rectangles (MBRs) of railway lines (rrlines) and 76999 MBRs of hypsography data (hypsogr). Data points are end points of hypsogr and obstacles are rrlines in Germany. In this way, data points and obstacles are adjacently deployed into the same geographical space and thus, simulate our scenario. Though we have used MBRs to represent obstacles, our algorithms support any arbitrary shaped polygons. We generate synthetic data sets U and Z using a uniform and Zipfian distribution, respectively. The data points and obstacles are indexed using separate R-trees in the database.

We consider 100 sample kOGNN queries and obtain the average experimental results. The query points are allowed to lie on the boundary of the obstacles but not inside the obstacles. We measure the CPU time and IO cost of the proposed algorithms using an Intel Core i7-2920XM quad-core CPU (2.50GHz) PC with 16GB RAM.

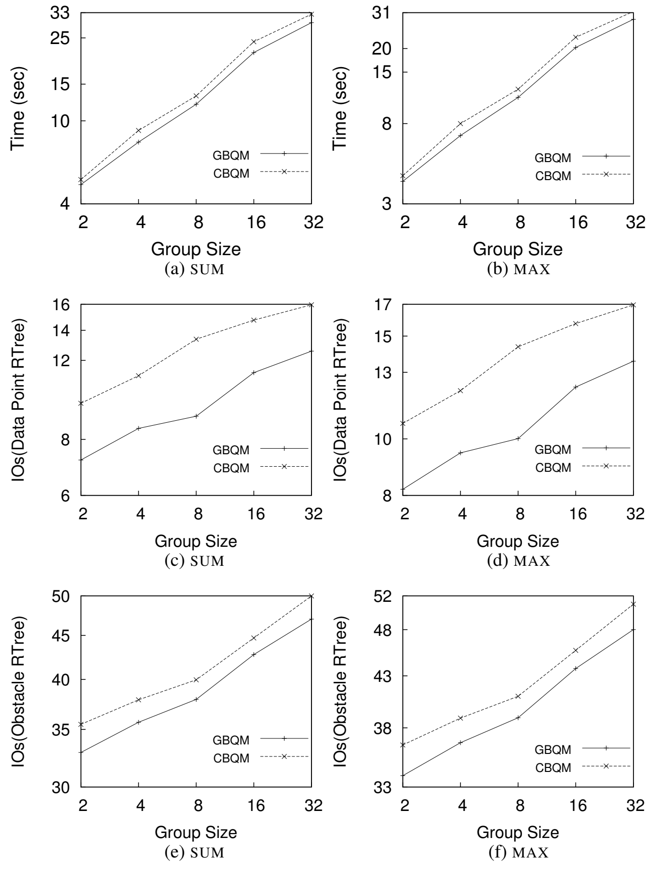


Figure 2: Effect of group size (data set Germany)

6.1 Effect of group size

Figure 2 shows that the performance of GBQM and CBQM degrade as the group size increases. The reason behind such an increase is expected: the increase in the group size increases the number of obstructed distance computations and hence increases more obstacle retrieval from the obstacle RTree. For both SUM and MAX, we observe that GBQM performs better than CBQM, e.g., GBQM requires on average 1.33 times lower IO access than CBQM for retrieving data points. We also observe in Figure 2 that MAX gives lower CPU time and IO access than those for SUM because the refined search area is smaller for MAX than SUM.

6.2 Effect of k and query rectangle area

Experimental results for both varying k and query rectangle area show similar behavior as the group size. The time, IO access of the data point R-tree and obstacle R-tree required by GBQM and CBQM for the aggregate function SUM and MAX increase with the increase of both k and the query rectangle area (not shown). We also observe that GBQM outperforms CBQM in terms of both time and IO access for different values of k and query rectangle area.

6.3 Effect of Cardinality Ratio

Experiments show that the CPU time and IOs for the obstacle RTree decreases with the increase of the ratio $|P|/|O|$ on dataset U (not shown). The reason behind this is, with the increase of cardinality, obstacle retrieval decreases as a consequences CPU time also decreases. On the other hand, we find in experiments that the IO accesses of the data point RTree remains almost constant with the increase of $|P|/|O|$ because, as the density increases, the range around the set of query points where the Euclidean neighbors or Euclidean GNN are found decreases.

7. CONCLUSION

We have developed the first comprehensive solution to find the k group nearest neighbors in presence of obstacles. We have proposed an efficient aggregate distance computation technique, and two OGNN algorithms, CBQM and GBQM. In our experiments, we have shown that our algorithms can compute OGNNs in real time. Our comparative analysis reveals that the CPU time and IO access of GBQM are always lower than those of CBQM for both SUM and MAX. In the future, we aim to protect user privacy [3, 5] while accessing OGNN queries.

8. REFERENCES

[1] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer, 1997.

[2] 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, 2011.

[3] T. Hashem and L. Kulik. Safeguarding location privacy in wireless ad-hoc networks. In *Ubicomp*, pages 372–390, 2007.

[4] T. Hashem, L. Kulik, and R. Zhang. Privacy preserving group nearest neighbor queries. In *EDBT*, pages 489–500, 2010.

[5] T. Hashem, L. Kulik, and R. Zhang. Countering overlapping rectangle privacy attack for moving knn queries. *Inf. Syst.*, 38(3):430–453, 2013.

[6] T. Lozano-Pérez and M. A. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Commun. ACM*, 22(10):560–570, 1979.

[7] S. Namnandorj, H. Chen, K. Furuse, and N. Ohbo. Efficient bounds in finding aggregate nearest neighbors. In *DEXA*, pages 693–700, 2008.

[8] D. Papadias, Q. Shen, Y. Tao, and K. Mouratidis. Group nearest neighbor queries. In *ICDE*, pages 301–312, 2004.

[9] W. Sun, C. Chen, B. Zheng, C. Chen, L. Zhu, W. Liu, and Y. Huang. Merged aggregate nearest neighbor query processing in road networks. In *CIKM*, pages 2243–2248, 2013.

[10] H. Xu, Z. Li, Y. Lu, K. Deng, and X. Zhou. Group visible nearest neighbor queries in spatial databases. In *WAIM*, pages 333–344, 2010.

[11] M. L. Yiu, N. Mamoulis, and D. Papadias. Aggregate nearest neighbor queries in road networks. *IEEE Trans. Knowl. Data Eng.*, 17(6):820–833, 2005.

[12] J. Zhang, D. Papadias, K. Mouratidis, and M. Zhu. Query processing in spatial databases containing obstacles. *International Journal of Geographical Information Science*, 19(10):1091–1111, 2005.