

M.SC. ENGG. THESIS

# Efficient Algorithms for Computing Optimal Meeting Points in the Obstructed Space

by  
Tahsina Hashem

Submitted to

Department of Computer Science and Engineering  
in partial fulfilment of the requirements for the degree of  
Master of Science in Computer Science and Engineering



Department of Computer Science and Engineering  
Bangladesh University of Engineering and Technology (BUET)  
Dhaka 1000

December 2017

*Dedicated to my loving parents*

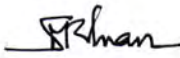
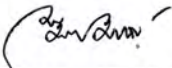
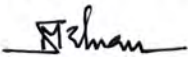
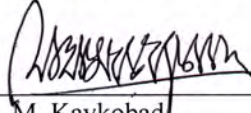
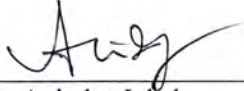
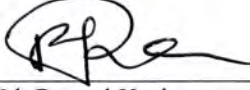
## AUTHOR'S CONTACT

---

Tahsina Hashem  
Assistant Professor  
Department of CSE  
Jahangirnagar University  
Email: tahsinahashem@gmail.com

The thesis titled “Efficient Algorithms for Computing Optimal Meeting Points in the Obstructed Space”, submitted by Tahsina Hashem, Roll No.: **0413052093 P**, Session April 2013, to the Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, has been accepted as satisfactory in partial fulfilment of the requirements for the degree of Master of Science in Computer Science and Engineering as to its style and contents. Examination held on December 3, 2017.

### **Board of Examiners (M. Sc. Engineering)**

1.   
\_\_\_\_\_  
Dr. M. Sohel Rahman  
Professor  
Department of CSE, BUET, Dhaka. **Chairman  
(Supervisor)**
2.   
\_\_\_\_\_  
Dr. Md. Mostofa Akbar  
Professor  
Department of CSE, BUET, Dhaka. **Member  
(Co-Supervisor)**
3.   
\_\_\_\_\_  
Dr. M. Sohel Rahman  
Head  
Department of CSE, BUET, Dhaka. **Member  
(Ex-Officio)**
4.   
\_\_\_\_\_  
Dr. M. Kaykobad  
Professor  
Department of CSE, BUET, Dhaka. **Member**
5.   
\_\_\_\_\_  
Dr. Anindya Iqbal  
Assistant Professor  
Department of CSE, BUET, Dhaka. **Member**
6.   
\_\_\_\_\_  
Dr. Md. Rezaul Karim  
Professor  
Department of CSE,  
University of Dhaka, Dhaka. **Member  
(External)**

## Candidate's Declaration

This is hereby declared that the work titled “Efficient Algorithms for Computing Optimal Meeting Points in the Obstructed Space” is the outcome of research carried out by me under the supervision of Dr. M. Sohel Rahman and Dr. Md. Mostofa Akbar, in the Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka - 1000. It is also declared that this thesis or any part of it has not been submitted elsewhere for the award of any degree or diploma.

*Tahsina Hashem*

---

Tahsina Hashem  
Candidate

# Acknowledgment

I would like to give my heart-felt thanks to my supervisor, Dr. M. Sohel Rahman, and to my co-supervisor, Dr. Md. Mostofa Akbar for their constant support, encouragements and guidance. I am very much grateful for their confidence on my capabilities, which always inspired me to produce high quality research. The long hours of fruitful discussions and arguments with my supervisor and co-supervisor helped me to develop myself as a creative researcher. This research would not have been possible without their patience and advice.

I would also want to thank the honorable members of my thesis committee for their valuable suggestions. I thank Dr. M. Kaykobad, Dr. Anindya Iqbal and specially the external member Dr. Md. Rezaul Karim.

I would like to express my sincere gratitude to Tanzima Hashem for helping me a lot in my thesis work. Special thanks to my parents, sisters, and my brothers for their support and encouragements that cheered me up at different stages of my M.Sc. Thesis. I am also grateful to my grandparents for their payers and wishes for my success of life.

Finally, I remain ever grateful to Almighty for His kindness that allows me to successfully complete this thesis.

# Abstract

In this thesis, we introduce an *Obstructed Optimal Meeting Point* (OOMP) query that enables a group of users to identify a meeting location with the minimum total travel distance in presence of obstacles. For example, a group of friends located at different locations in the park or in a city center may want to meet at a point that minimizes the total distance of the group members. However, in the park or city center they may have to face many obstacles like trees or lakes and buildings in their walking paths. In recent years, the problem of finding optimal meeting point (OMP) has been addressed in the Euclidean space and road networks which ignores the presence of obstacles. We show that the problem of finding OMP in the obstructed space is NP-hard. We introduce heuristic algorithm for processing an OOMP query. Processing an OOMP query in the obstructed space is an exhaustive search, as the search space is infinite and filled with obstacles. To identify the optimal meeting point, computing the total obstructed distance for every point in the search space would incur extremely high processing overhead as finding the obstructed distance between two locations is an expensive computation. Thus, the major challenges for an OOMP query is to refine the search space and compute the total obstructed distance with reduced processing overhead. We exploit geometric properties and hierarchical structure to develop techniques to refine the search space. In addition, we develop efficient technique to compute the total obstructed distance. Our technique reduces the number of obstacles retrieved from the database and does not retrieve the same obstacle multiple times from the database to compute multiple individual obstructed distances required for computing a total obstructed distance. The query processing overhead increases with the increase of the number of the group members, obstacles and the search space. To further decrease the processing overhead, we develop another heuristic algorithm for processing an OOMP query in real time by sacrificing accuracy. We evaluate the efficiency and effectiveness of our algorithms using real datasets and present a comparative analysis among our proposed algorithms.

# Table Of Contents

<i>Board of Examiners (M. Sc. Engineering)</i>	ii
<i>Candidate's Declaration</i>	iii
<i>Acknowledgment</i>	iv
<i>Abstract</i>	v
<b>1 Introduction</b>	<b>1</b>
1.1 Research Problem . . . . .	2
1.1.1 An Obstructed Space . . . . .	4
1.1.2 Minimum and Maximum Total Obstructed Distance . . . . .	4
1.1.3 Obstructed Optimal Meeting Point ( <i>OOMP</i> ) Queries . . . . .	6
1.1.4 Hardness . . . . .	8
1.2 Solution Overview . . . . .	9
1.2.1 Minimum and Maximum Total Obstructed Distance Computation . . . . .	9
1.2.2 <i>OOMP</i> Algorithms . . . . .	10
1.3 Contributions . . . . .	11
1.4 Outline . . . . .	11
<b>2 Related Work</b>	<b>12</b>
2.1 Optimal Meeting Point Queries in the Euclidean Space . . . . .	12
2.2 Optimal Meeting Point Queries in Road Networks . . . . .	13
2.3 Obstructed Path Problems in Computational Geometry . . . . .	13
2.4 Spatial Queries in the Obstructed Space . . . . .	14
2.4.1 Obstructed Nearest Neighbor ( <i>ONN</i> ) Queries . . . . .	14

2.4.2	Obstructed Reverse Nearest Neighbor (ORNN) Queries . . . . .	15
2.4.3	Continuous Obstructed Nearest Neighbor (CONN) Queries . . . . .	15
2.4.4	Moving Nearest Neighbor Queries . . . . .	16
2.4.5	Obstructed Group Nearest Neighbor (OGNN) Queries . . . . .	16
2.4.5.1	Aggregate Obstructed Distance from Multiple Points to a Single Point	16
2.4.5.2	OGNN query vs. OOMP query . . . . .	19
2.4.6	Visible $k$ Nearest Neighbor ( $k$ VNN) Queries . . . . .	19
2.4.7	Group Visible Nearest Neighbor (GVNN) Queries . . . . .	20
2.5	Indexing Techniques for Space Partitioning . . . . .	20
2.5.1	R-tree Indexing Structure . . . . .	21
<b>3</b>	<b>Obstructed Distance Computation</b>	<b>24</b>
3.1	Visibility Graph . . . . .	24
3.1.1	Incremental Visibility Graph Computation Strategy . . . . .	25
3.2	Obstructed Distance between two Points . . . . .	26
3.3	Minimum and Maximum Total Obstructed Distance Computation . . . . .	28
3.3.1	Minimum Obstructed Distance Computation from a Point to a Square . . . . .	29
3.3.2	Minimum Total Obstructed Distance Computation from Multiple Points to a Square . . . . .	32
3.3.2.1	Correctness of Algorithm CompMinTotalObsDist . . . . .	35
3.3.2.2	Time Complexity of Algorithm CompMinTotalObsDist . . . . .	36
3.3.3	Maximum Total Obstructed Distance Computation . . . . .	37
3.3.3.1	Time Complexity of Algorithm CompMaxTotalObsDist . . . . .	41
<b>4</b>	<b>Our Algorithms</b>	<b>42</b>
4.1	Hierarchical Algorithm . . . . .	42
4.1.1	Time Complexity of Hierarchical Algorithm . . . . .	51
4.2	Grid Algorithm . . . . .	52
4.2.1	Time Complexity of Grid Algorithm . . . . .	58
<b>5</b>	<b>Experiments</b>	<b>59</b>
5.1	Comparison of OOMP Algorithms . . . . .	60
5.1.1	Effect of Group Size . . . . .	60



5.1.2	Effect of Query Rectangle Area . . . . .	62
5.1.3	Effect of the Length of Square Grid Cell . . . . .	63
5.2	Comparative Analysis . . . . .	65
<b>6</b>	<b>Conclusion</b>	<b>66</b>
6.1	Contribution . . . . .	66
6.2	Future Work . . . . .	67
	<b>References</b>	<b>69</b>

# List of Figures

1.1	An example of OOMP query in the obstructed space . . . . .	2
1.2	Application Scenarios of OOMP query . . . . .	3
1.3	Obstructed Distance between two points . . . . .	5
1.4	Minimum obstructed distance from query points to a square . . . . .	6
1.5	Maximum obstructed distance from query points to a square . . . . .	7
1.6	An example of <i>OOMP</i> query . . . . .	7
2.1	Single Point Based Aggregate Obstructed Distance (SPAOD) computation . . . . .	17
2.2	Multi Point Based Aggregate Obstructed Distance (MPAOD) . . . . .	18
2.3	OGNN query vs. OOMP query . . . . .	19
2.4	Example of R-tree . . . . .	21
2.5	A best first search example in R-tree . . . . .	22
3.1	Obstructed distance computation through visibility graph . . . . .	25
3.2	Obstructed distance computation between $p$ and $q$ . . . . .	26
3.3	Minimum obstructed distance computation example . . . . .	30
3.4	Minimum total obstructed distance computation . . . . .	34
3.5	Computation of maximum obstructed distance from a query point to 4 Lines . . . . .	38
3.6	Computation of maximum obstructed distance from a quadrant to a query point . . . . .	39
4.1	Initial bounded search space, $R_{sq}$ of hierarchical algorithm . . . . .	48
4.2	Recursively refining the initial search space of hierarchical algorithm . . . . .	49
4.3	Grid representation of bounded square region, $R_{sq}$ . . . . .	53
4.4	Best first search for finding optimal meeting vertex of three query points . . . . .	57

5.1	Effect of group size . . . . .	61
5.2	Effect of group size on approximation error . . . . .	62
5.3	Effect of query rectangle area . . . . .	63
5.4	Effect of query rectangle area on approximation error . . . . .	63
5.5	Effect of square length . . . . .	64
5.6	Effect of square length on approximation error . . . . .	64

# List of Tables

2.1	BFS in R-tree . . . . .	22
3.1	Notations and their meanings . . . . .	28
4.1	Notations and their meanings . . . . .	43
5.1	Experiment setup . . . . .	59

# List of Algorithms

1	CompMinObsDist( $q, R, LVG$ ) . . . . .	29
2	CompMinTotalObsDist( $Q, R, RT_O, LVG$ ) . . . . .	33
3	CompMaxTotalObsDist( $Q, R, RT_O, LVG$ ) . . . . .	37
4	Hierarchical_Algorithm( $Q, RT_O$ ) . . . . .	45
5	FindBoundedRegion( $Q, RT_O$ ) . . . . .	46
6	Grid_Algorithm ( $Q, RT_O$ ) . . . . .	55

# Chapter 1

## Introduction

Finding an Optimal Meeting Point (OMP) is a fundamental problem in the field of spatial query processing. For example, a group of friends located at different locations in the park or in a city center may want to meet at a point that minimizes the total distance of the group members. However, in the park or city center they may have to face many obstacles like trees or lakes and buildings in their walking paths. In recent years, the problem of OMP has been addressed in the Euclidean space [1–5] and Road networks [5–7] that ignore the presence of obstacles. In this thesis, we introduce an Obstructed Optimal Meeting Point (OOMP) query that enables a group to identify the location of the meeting point that minimizes the total obstructed distance.

Processing an OOMP query in the obstructed space is an exhaustive search, as the search space is infinite and filled with obstacles. To identify the optimal meeting point, computing the total obstructed distance for every point in the search space would incur extremely high processing overhead as finding the obstructed distance between two locations is an expensive computation. Thus, the efficiency of an OOMP algorithm depends on (i) the number of meeting points that needs to be considered in the search space and (ii) the efficiency of the algorithm to compute the total obstructed distance. The smaller number of considered locations in the search space also decreases the number of obstructed distance computations.

In this thesis, we exploit geometric properties and hierarchical structure to develop techniques to refine the search space. In addition, we develop efficient technique to compute the total obstructed distance. The cost of processing overhead depends on the number of obstacles considered while computing the

total obstructed distance. The less the number of obstacles retrieved from the database, the more we can minimize the processing overhead. Our technique reduces the number of obstacles retrieved from the database and does not retrieve the same obstacle multiple times from the database to compute multiple individual obstructed distances required for computing a total obstructed distance.

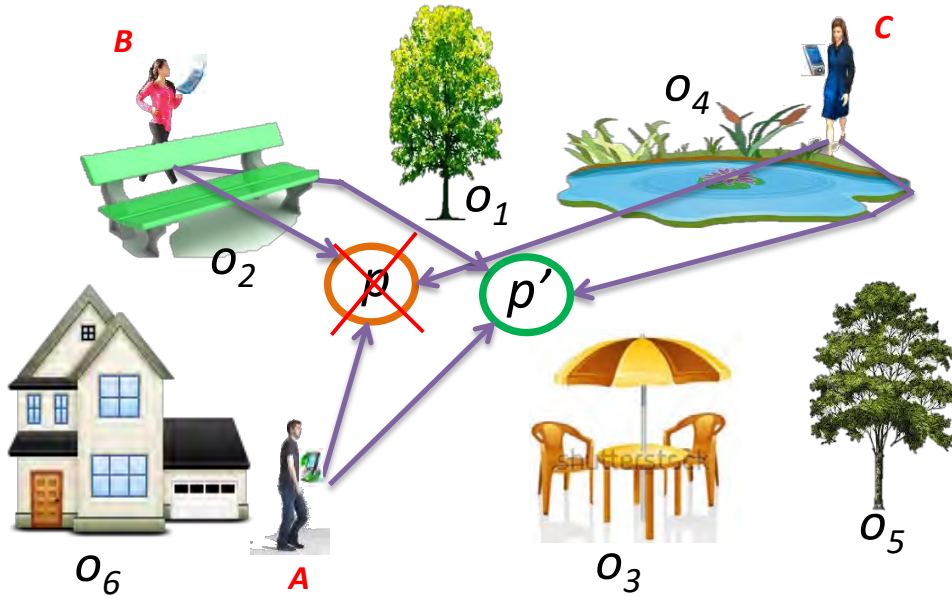


Figure 1.1: An example of OOMP query in the obstructed space

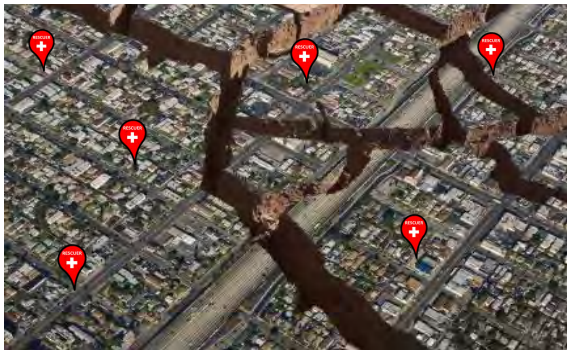
## 1.1 Research Problem

The Obstructed space is different from the Euclidean space and the road network space. In the Euclidean space, user is allowed to move freely along the space, and in the network space, although user has to follow the network structure but can move freely through the route of the network. However, in the obstructed space, user cannot go through obstacles. The straight line distance between any two points is blocked in presence of obstacles. Hence, the *obstructed distance* between two points is the length of the shortest path connecting the two points, without crossing the interior of any obstacles.

Consider an example scenario of Figure 1.1. Suppose a group of pedestrians  $A$ ,  $B$  and  $C$  in the park want to meet at a location that minimizes their total travel distance. Now if we use Euclidean distances to find the optimal meeting point then the optimal meeting point will be located at location

$P$  since Euclidean distance is measured with the length of the straight lines between points. However, in reality, there are many obstacles like trees, benches, lakes that can obstruct the travel path of pedestrians. For example pedestrian  $B$  cannot walk through the fence or pedestrian  $C$  can not walk through the lake. Thus pedestrians have to walk by avoiding obstacles. If we consider obstructed distance then the optimal meeting point will be at location  $P'$ . Since,  $P'$  minimizes the total obstructed distance of group members.

Now-a-days there is a continuous advancement in the usage of smartphones and other GPS enabled devices. Furthermore location based social networks allows group of friends to share their locations with each others and request a location-based service for the group. Thus using our proposed solutions of OOMP query via location aware mobile devices, a group of pedestrians can find the meeting point in the obstructed space efficiently.



(a) Earthquake Effect



(b) Battlefield

Figure 1.2: Application Scenarios of OOMP query

In addition, this problem is also related to path planning and motion control in robotics, computational geometry, complex spatial data mining involving obstacles etc. Figure 1.2(a) and 1.2(a)(b) shows two application scenario of OOMP query. The left figure shows an example scenario of an earthquake effect. Here emergency help is required to locate survivors from the obstructed area as early as possible. A number of rescuers are needed to meet together in an optimal point to facilitate the excavation. The right figure shows an example scenario of a battlefield. Here also several tanks or soldiers can move in any direction if no obstacle obstructs their path. A group of soldiers/tanks located in different places need to meet urgently in an optimal meeting point to launch an attack



together.

A number of research has been conducted in the area of finding Optimal Meeting Points on the Euclidean space and the Road Network space [1–7]. None of the algorithms consider the presence of obstacles among the locations of group members. Thus, we are the first to address the problem of finding an OMP query in the obstructed space. Research works involving obstacles are shortest path queries [8], range and nearest neighbor (NN) queries[9], Obstructed Reverse Nearest Neighbor (ORNN) queries [10], Continuous NN queries[11] , moving NN queries[12], Visible  $k$  nearest neighbor ( $VkNN$ ) queries[13–15] and Obstructed Group Nearest Neighbor (OGNN) queries[16].

In the following sections we formally discuss the obstructed space, the minimum and maximum total obstructed distance and obstructed optimal meeting point queries. We also discuss basic ideas that we are going to use throughout the thesis.

### 1.1.1 An Obstructed Space

An obstructed space consists of obstacles like buildings, lakes, roads for vehicles, trees etc. and the query points i.e., the group of people introducing the query. Obstructed space is different from the Euclidean space and the network space. As we already discussed, in the Euclidean space the distance is measured as the length of the straight line between two points., and in the road network space the distance is measured as the length of the shortest path between two points in the given network structure. As a result movement is permitted in a predefined network structure. On the other hand, 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 considering the obstacles. Thus, in the road network, we define the roads that can be travelled and in the obstructed space we define the obstacles that should be avoided.

### 1.1.2 Minimum and Maximum Total Obstructed Distance

The obstacle path problem is a popular topic in Robotics, Computational Geometry, GIS, Game Planning etc. Obstructed path is the shortest path among two points  $p_1$  and  $p_2$  in the presence of a set of obstacles  $O$ . The obstacles are non-overlapping  $2D$  polygons and the shortest path among the two points do not cross the interior of any obstacle. The shortest path distance connecting two

objects without crossing any obstacle in an obstructed space (space consisting of obstacles/polygon) is called the obstructed distance  $dist_O(p_1, p_2)$ , between the two points  $p_1$  and  $p_2$ . In Figure 1.3 the obstructed distance  $dist_O(p_1, p_2)$ , between the two points  $p_1$  and  $p_2$  is shown by the solid line. The shaded polygons are obstacles in this obstructed space. The Euclidean distance  $dist_E(p_1, p_2)$ , between the two points  $p_1$  and  $p_2$  is also shown by a dashed line.

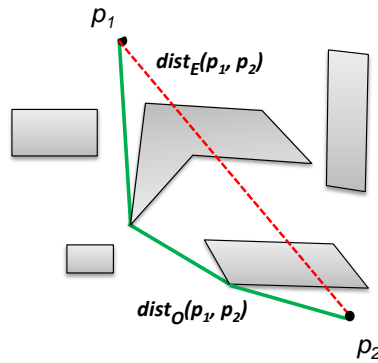


Figure 1.3: Obstructed Distance between two points

In the Euclidean space, the minimum distance from a query point  $q_i \in Q$  to a square  $R$ ,  $distMin_E(q_i, R)$  is the straight line distance from the query point to the nearest point on the boundary lines of the square and the maximum distance from a query point  $q_i \in Q$  to a square  $R$ ,  $distMax_E(q_i, R)$  is the straight line distance from the query point to the furthest point on the boundary lines of the square. In presence of obstacles these distances might not exact. Any point on the boundary lines of the square might give the minimum and maximum obstructed distance. However, there exists infinite number of points on the boundary lines of the square. It is quite infeasible to examine each of these points and find out the minimum and maximum obstructed distance. Thus, we need to develop pruning techniques to refine the search space.

In Figure 1.3, the dotted line shows the minimum Euclidean distance from a query point  $q_i \in Q$  to the square  $R$ ,  $distMin_E(q_i, R)$  and the solid line shows minimum obstructed distance  $distMin_O(q_i, R)$  from the query point  $q_i \in Q$  to the square  $R$ .

The lower bound of minimum total obstructed distance from a square to all the query points ( let  $n$

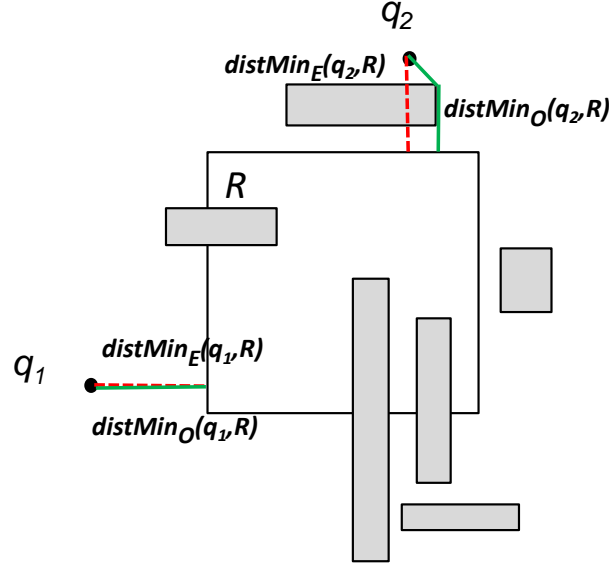


Figure 1.4: Minimum obstructed distance from query points to a square

be the number of query points),  $distMin_{TO}(Q, R)$  can be defined as:

$$distMin_{TO}(Q, R) = \sum_{i=1}^n distMin_O(q_i, R) \quad (1.1)$$

The Figure 1.5 shows the maximum Euclidean distance,  $distMax_E(q_i, R)$  and maximum obstructed distance,  $distMax_O(q_i, R)$ .

Similarly the upper bound of maximum total obstructed distance from a square to all the query points,  $distMax_{TO}(Q, R)$  can be defined as:

$$distMax_{TO}(Q, R) = \sum_{i=1}^n distMax_O(q_i, R) \quad (1.2)$$

Obstacles can be in any shape (e.g., triangle, pentagon, etc.), we assume it is a rectangle in this research. We store all our obstacles and data points in a spatial database indexed by R-tree. We use an R-tree based indexing method for our searching and retrieval algorithms, though any kind of indexing is applicable for our algorithms.

### 1.1.3 Obstructed Optimal Meeting Point (OOMP) Queries

In the obstructed space there might be infinite points i.e.  $p_1, p_2, \dots, p_m, \dots, p_\infty$ . The total obstructed distance between any point  $p_m$  from the obstructed space to all the query points can be defined as:

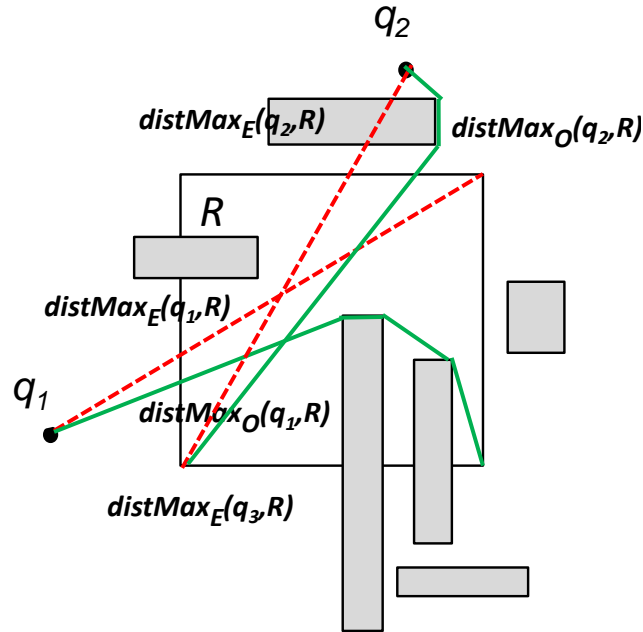


Figure 1.5: Maximum obstructed distance from query points to a square

$$dist_{TO}(p_m, Q) = \sum_{i=1}^n dist_O(p_m, q_i) \tag{1.3}$$

**Definition 1.1.1. (OOMP)** A point  $p_{oomp}$  is the obstructed optimal meeting point OOMP, if the total obstructed distance of the point  $p_{oomp}$  to all the query points is the smallest among all the points in the obstructed space according to the following Equation 1.4:

$$dist_{TO}(p_{oomp}, Q) = \min_{i=1}^{\infty} dist_{TO}(p_i, Q) \tag{1.4}$$

The following figure 1.6 shows an example of OOMP queries in the obstructed space.

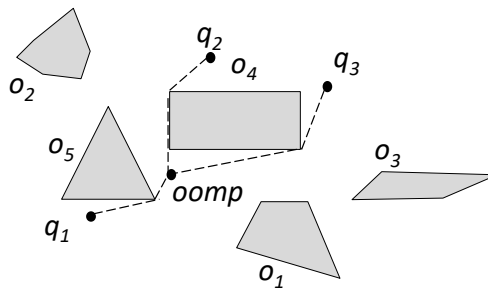


Figure 1.6: An example of OOMP query

### 1.1.4 Hardness

Finding an optimal meeting point (OMP) in the obstructed space is very much hard, since in the search space there are infinite number of points exist. We show that finding an optimal meeting point in the obstructed space is an NP-hard problem.

**Theorem 1.1.1.** *The problem of computing optimal meeting point in the obstructed space is NP-hard.*

**Proof** We reduce a known NP-hard problem, Weber [3] problem to our problem of OOMP query. Weber problem is a simple minimum facility location [17] problem and it is proved in [17] that the weber problem is NP-hard. It requires finding a single facility point in the plane that minimizes the sum of the transportation costs from the facility point to a number of destination points, where different destination points are associated with different costs per unit distance. If the transportation costs per unit distance are the same for all destination points, then weber problem is called geometric median problem or 1-median problem. Thus weber problem finds the optimal facility point  $p$  of a number of destination points  $D = \{d_1, d_2, \dots, d_n\}$  in the Euclidean space, that minimizes the following weighted sum:

$$\operatorname{argmin}_p \sum_{i=1}^n \operatorname{dist}_E(p, d_i) \quad (1.5)$$

Here,  $\operatorname{dist}_E(x, y)$  represents the Euclidean distance or transportation cost between the two points  $x$  and  $y$  in the Euclidean space and  $\operatorname{argmin}$  represents the value of the argument  $p$  which minimizes the sum. Whereas in OOMP query, we have to find out the location of optimal meeting point  $p'$  of a number of query points  $Q = \{q_1, q_2, \dots, q_n\}$  that minimizes the total obstructed distance of the query points in the obstructed space.

$$\operatorname{argmin}_{p'} \sum_{i=1}^n \operatorname{dist}_O(p', q_i) \quad (1.6)$$

Here,  $\operatorname{dist}_O(x, y)$  represents the obstructed distance between the two points  $x$  and  $y$  in the obstructed space (which is definitely harder than finding Euclidean distance ) and  $\operatorname{argmin}$  represents the value of the argument  $p'$  which minimizes the sum. If the destination points  $D$ , represent the query points  $Q$ , the single facility represents the location of optimal meeting point and the weighted sum represents the sum of obstructed distances from the optimal meeting point to all the query points, then the Weber problem can be mapped to finding the location of optimal meeting point in the obstructed

space which minimizes the total obstructed distances of all the query points, which is exactly the goal of our OOMP query.

## 1.2 Solution Overview

In the real world there are large number of obstacles in a pedestrian's walking path, finding the exact optimal meeting point in real time is a major challenge. The smaller number of locations that an OOMP algorithm requires to consider for identifying the actual OOMP, the more efficient the algorithm is as for each location, an OOMP algorithm computes the total obstructed distance from the candidate location to the locations of pedestrians. Two heuristic algorithms are developed for processing OOMP queries. The obstacles are stored on the database with indexing. A number of pruning techniques have been developed to refine the infinite search space and thereby reduce the query processing overhead. In addition, we present efficient algorithms to compute minimum and maximum total obstructed distance from a square to all pedestrians, which is a key component to efficiently evaluate the OOMP. Considering all obstacles for computing distance every time is not a feasible solution. The base idea of our minimum and maximum total obstructed distance computation technique is to consider only those obstacles, which are required for distance computation and to avoid retrieving the same obstacles multiple times.

In Section 1.2.1 and Section 1.2.2, we give a short overview of our proposed solutions for computing maximum and minimum total obstructed distance from a square to set of pedestrians and the OOMP query evaluation.

### 1.2.1 Minimum and Maximum Total Obstructed Distance Computation

There are algorithms [16, 18] for computing total obstructed distance from a single data point to multiple query points. But no algorithm exists for computing minimum and maximum total obstructed distance from a set of points to a square. However these two distances are playing major role to refine the infinite search space and to reduce query processing overhead for finding the OOMP.

To compute minimum total obstructed distance from a set of query points to a square, we can apply

our developed minimum obstructed distance computation algorithm by considering the square and one query point ( $q \in Q$ ) at a time from the set of query points ( $Q$ ), then applying summation operation in the results. However, this technique involves high IO access. Same obstacle may be considered multiple times for different query points in the group ( $Q$ ), which is not desired. Thus we also develop algorithm for computing minimum total obstructed distance of a set of query points from a square. The key idea behind this algorithm is, when computing minimum total obstructed distance between a square and a set of query points  $Q$ , we retrieve obstacles having distance equal to a threshold with respect to the center point of the square. The algorithm does not retrieve any unnecessary obstacle, which are not required for computing minimum total obstructed distances and also does not retrieve the same obstacle multiple times from the database. Thus the algorithm reduces query processing overhead significantly.

There is algorithm [11] for computing maximum obstructed distance from a point to a straight line. In our *OOMP* algorithm, we divide the search space hierarchically into four quadrants. The quadrant is a square which consists of four lines. Applying the algorithm [11] for four line segments, we have calculated the maximum obstructed distance from a query point to a square quadrant and then summing up we get the total maximum obstructed distance from a quadrant to all query points.

## 1.2.2 OOMP Algorithms

We have developed two heuristic algorithms for processing OOMP queries. We call our first heuristic algorithm as the hierarchical algorithm because it refines the search space in a recursive manner. The search starts by considering the total space. In every iteration of the search, the search space is divided into four quadrants, and the quadrant that has the highest probability to include the OOMP is considered in the next iteration. The quadrants that cannot have the OOMP are gradually pruned by exploiting geometric properties. The search terminates when the approximate meeting point is identified.

The query processing overhead increases with the increase of the number of the group members, obstacles and the search space. In practical applications, it is usual to sacrifice the accuracy slightly if the OOMP query can be processed in real time. Therefore, the second heuristic algorithm is developed. The second heuristic algorithm finds the meeting point with less accuracy compared to

first heuristic algorithm but incurs reduced processing overhead and computation time. We call this heuristic algorithm as the grid algorithm since it divides the refined search space into a grid.

## 1.3 Contributions

We summarize our key contributions as follows:

- We introduce *obstructed optimal meeting point* (OOMP) queries in an obstructed space.
- We develop efficient algorithms to compute minimum and maximum total obstructed distance between a square and a set of query points.
- We develop two heuristic algorithms to approximate an OOMP.
- We perform experimental analysis of our developed algorithms with real datasets and show that our proposed algorithms can determine OOMP with reduced time and space overhead.

## 1.4 Outline

The rest of the thesis is organized as follows:

In Chapter 2, We study existing research related to different types of OMP queries in spatial database along with some spatial indexing technique.

In Chapter 3, We explain the algorithms for computing maximum and minimum total obstructed distance from a square to the query points.

In Chapter 4, We describe our two algorithms to evaluate *obstructed optimal meeting point* (OOMP).

In Chapter 5, We implement our algorithms and show some experimental results using real datasets.

In Chapter 6, summary and outcome of the thesis with possible future directions are described.



## Chapter 2

# Related Work

In this chapter, we discuss existing work of optimal meeting point queries and spatial queries in the obstructed space. First of all in Sections 2.1 and 2.2, we review the work related to optimal meeting point queries in the Euclidean space and in the road network space. Existing obstacle path problems in the area of computational geometry and data clustering literature is illustrated in Sections 2.3 and 2.4. In Section 2.5, a short overview of spatial queries in the obstructed space is presented. The chapter concludes by analyzing indexing techniques for storing and accessing data from the database.

### 2.1 Optimal Meeting Point Queries in the Euclidean Space

Optimal meeting point (OMP) queries were first introduced by Cooper for the Euclidean space in 1960 [3]. They defined the OMP query as *Weber Problem* and the OMP is called geometric median of query point set. Their goal is to minimize aggregate euclidean distance for all users. Beside this, a number of researches [1, 2, 4] have been conducted in this area. Later Cooper [3] express the OMP problem as the problem of minimizing the weighted sums of powers of the Euclidean distances. Further Chen [2] generalized the weber problem for solving radial cost functions. Finally, researchers show that gradient descent methods [19–22] are the best suited to solve the Weber Problem without the worry of being stuck at local minimum. Recently, different variety of OMP queries have been studied [5]. They also proposed gradient descent solution for weighted OMP queries and identified the best algorithm for particular types of OMP queries.

## 2.2 Optimal Meeting Point Queries in Road Networks

Researchers considered OMP query on road network space also. First initiative was taken in 2010 [23]. They proved that that an OMP must exist among the *split points* on a road network. For a point  $p$  on a road network, its split point on edge  $(u, v)$  is defined to be the point  $x$  such that  $d(\overline{p, u}) + d(u \sim x) = d(\overline{p, v}) + d(v \sim x)$ , where  $d(\overline{a, b})$  represents the length of the shortest path between points  $a$  and  $b$  and  $d(a \sim b)$  represents the line segment of an edge with endpoints  $a$  and  $b$ . But their proposed required search space is  $|Q| \cdot |E|$ , which is huge. Yan [6] proposed a new baseline algorithm for min-sum OMP query which reduces the search space to  $|V| + |Q|$ . They also gave an effective two-phase convex-hull-based pruning technique to further prune the search space. Most importantly, they developed an extremely efficient greedy algorithm to find a high-quality near-optimal meeting point instead of an exact OMP, which is orders of magnitude faster than the exact OMP algorithms. But these two methods do not guarantee optimal results. Further, Yan [5] proposed R-tree based branch and bound algorithm both for min-sum and min-max OMP queries. They suggest two pruning techniques namely Euclidean distance bound and threshold algorithm. Recently Tiwari [7] proposed some grid based algorithms to find  $k$  optimal meeting points on road network databases.

## 2.3 Obstructed Path Problems in Computational Geometry

Path problems in the obstructed space are studied in computational geometry [24]. Most solutions are based on visibility graph and visibility polygon [25–30]. It is [31] proved that the shortest path between source and destination point lies in the visibility graph and can be computed by any conventional shortest path algorithms [32, 33]. The literatures in computational geometry rely on preprocessing and does not prune any obstacle. Since, spatial database applications may require update of spatial data any time, preprocessing will not give exact result set. On the other hand, considering all the obstacles for each query point is costly. For very large databases, real time query processing is almost impossible for query processing overhead.

## 2.4 Spatial Queries in the Obstructed Space

However, none of the algorithms discussed in Sections 2.1 and 2.2, considers the presence of obstacles in the space. Thus, we are the first to address the problem of finding OMP query in the obstructed space. Research works involving obstacles are shortest path queries [33] in the obstructed space, range and nearest neighbor (NN) queries [34] in the obstructed space, obstructed reverse nearest neighbor (ORNN) queries [10], continuous obstructed NN queries (CONN) [11], moving NN queries [12] and obstructed group NN queries (OGNN)[18]. These spatial queries are illustrated in the following sections.

### 2.4.1 Obstructed Nearest Neighbor (ONN) Queries

An obstructed nearest neighbor query was introduced by Dimitris et al. [34]. They developed algorithms for range search, nearest neighbors, e-distance joins and closest pairs, considering that both data objects and obstacles are indexed by R-trees. An obstacle range (OR) query returns all the data points from the given set of points,  $P$  those are within the given range of obstructed distance from query point. An obstructed nearest neighbor (ONN) query returns the data points from the set  $P$ , that have the smallest obstructed distances from query point. An obstacle e-distance join (*ODJ*) query returns all entity pairs from the given two set of entities, where the distance between the returned pair of points is within the range of given distance  $e$ . The closest pair query returns the closest data points between two data points set, which have the smallest obstructed distance.

An efficient obstructed nearest neighbor algorithms are proposed [35]. The obstructed distance is computed incrementally using a visibility graph with only the core obstacles. They also proposed efficient algorithms for finding the candidate data points pruning out a lot of unnecessary data points.

Gu and Yu developed algorithms [36] for finding obstructed group nearest neighbors using a grid-partition index combined with the obstructed voronoi diagram instead of R-tree based index. The obstructed voronoi diagram is precomputed, thus it can not handle dynamically updated data points. It can only handle static objects, but the overall running cost is reduced . They also introduced a new concept named obstructed bisector and also proved that the nearest neighbor searching in this type of grid-partition based indexing is independent of the query point  $q$ . This paper presents three

types of pruning heuristics pruning by no obstacles, pruning by cell border *min* and pruning by cell border *max* while computing the obstructed distance.

### 2.4.2 Obstructed Reverse Nearest Neighbor (ORNN) Queries

Another similar type of query in an obstructed space is obstructed reverse nearest neighbor query, given a dataset  $P$  and a query point  $q$ , an ORNN query returns all the points in  $P$ , that have  $q$  as their nearest neighbor. In [10], the first approach was proposed for answering ORNN which follows a filter-refinement framework and requires no preprocessing and enables effective pruning heuristics. They also introduced a novel boundary region concept. This paper also presents an idea to compute the obstructed distance between two points  $p$  and  $q$  incrementally using previous shortest path calculation.

### 2.4.3 Continuous Obstructed Nearest Neighbor (CONN) Queries

Continuous obstructed nearest neighbor (CONN)[11, 37] query returns nearest neighbors of all the points in a given query line (the trajectory segment through which a client is moving ) in a two-dimensional (2D) space. It is assumed that no obstacle intersects the query line. They perform only a single query over the query line segment and process the relevant data points and obstacles via the concept of control points and quadratic-based split point computation approach. They propose two different approach to continuous obstructed  $k$  nearest neighbor and trajectory obstructed  $k$  nearest neighbor (TO $k$ NN) to compute the  $k$ NNs for each point along with an arbitrary trajectory.

They introduce the concept of *control points* that simplifies the computation and comparison of the obstructed distance between two objects. Given  $p$ : a point of interest,  $O$ :an obstacle set, and an interval  $R$  over the query line segment, a point  $cp$  is the control point of  $p$  over  $R$ , iff (i) the shortest path from  $p$  to any point on  $R$  passes through  $cp$ ; and (ii)  $cp$  is visible to every point on  $R$ . They propose a quadratic-based method to form *split points*, by solving quadratic inequalities. In order to find CONNs, they also developed algorithm to compute maximum obstructed distance from a point to a straight line segment.

### 2.4.4 Moving Nearest Neighbor Queries

Another similar type of query is moving  $k$  nearest neighbor query. Here the query object has no specified trajectory. Li and Gu developed algorithms [12] for solving this query. In [38], moving  $k$ -NN query is solved based on a safe-region concept called the  $V^*$ -Diagram. Our proposed obstructed optimal meeting point queries consider only static objects. Hence it differs from moving  $k$  nearest neighbor queries.

### 2.4.5 Obstructed Group Nearest Neighbor (OGNN) Queries

Obstructed Group Nearest Neighbor (OGNN) Query, returns the optimal meeting location from the given set of point of interests (POIs), like a restaurant, movie theatre, shopping mall etc. that minimizes the total/maximum travel distance of all the group members in presence of obstacles such as buildings and lakes. OGNN Query is introduced by Sultana et al. [16]. They developed two algorithms, Centroid Based Query Method (CBQM) and Group Based Query Method (GBQM). GBQM incrementally retrieves Euclidean Group Nearest Neighbors (GNNs) and refines the search space by exploiting the fact that the obstructed distance is always greater or equal to the Euclidean distance between two points. The search for OGNNs stops when the aggregate distance for a retrieved GNN becomes less or equal to the Euclidean aggregate distance of the last retrieved GNN. On the other hand, CBQM incrementally retrieves Euclidean NNs with respect to the centroid of the query points and refines the search space based on geometric properties. The search space becomes smaller with the retrieval of POIs and the search terminates when the POIs inside the refined region have been retrieved.

#### 2.4.5.1 Aggregate Obstructed Distance from Multiple Points to a Single Point

In order to minimize the aggregate obstructed distance of the group members, Sultana [16] also developed two efficient algorithms, Single Point Aggregate Obstructed Distance (SPAOD) and Multi Point Aggregate Obstructed Distance (MPAOD) for computing aggregate obstructed distance from a single point to multiple query points.

Considering all obstacles every time for distance computation is not a feasible solution. SPAOD does not retrieve the same obstacle multiple times but SPAOD may retrieve some additional obstacles

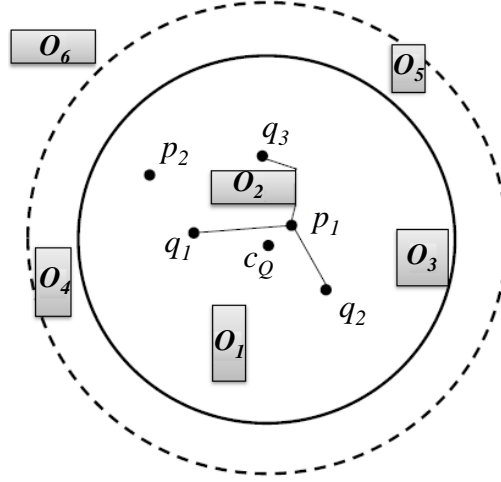


Figure 2.1: Single Point Based Aggregate Obstructed Distance (SPAOD) computation

that are not required for the aggregate obstructed distance computation. In addition, SPAOD reuses already computed obstructed shortest paths among different nodes in the visibility graph when computing the obstructed shortest path distances for subsequent data points.

Figure 2.1 shows an example of calculating the aggregate obstructed distance between data point  $p_1$  and query point set  $Q = \{q_1, q_2, q_3\}$ . SPAOD initially retrieves obstacles within the range,  $t = \max_{i=1}^3 (dist_E(c_Q, q_i) + dist_E(p_1, q_i))$  from the geometric centroid  $c_Q$  of  $Q$ , where  $t$  is the threshold. Then the algorithm compute obstructed distance of each query point  $q_i \in Q$  from  $p_1$ . If  $dist_O(p_1, q_i) \leq t$  for all  $q_i \in Q$ , then the algorithm stops retrieving obstacles. Because, according to Gao et al. [10] these distances are the real obstructed distances. Thus the algorithm stops retrieving obstacles and applying the aggregate function SUM or MAX, we get the aggregate obstructed distance  $dist_{AO}(p_1, Q)$  from  $p_1$  to  $Q$ . In this way, SPAOD retrieve obstacles incrementally and simultaneously computes obstructed distances between nodes in local visible graph,  $LVG$ . With the incremental retrieval of obstacles, the algorithm gradually update the local visibility graph. Whenever, a new data point  $p_2$  comes, SPAOD retrieves two new obstacles  $o_4$  and  $o_5$ , but it is not retrieving the previous set of obstacles again, thus SPAOD reuses the obstructed distance computations.

On the other hand, MPAOD does not reuse already computed obstructed shortest path distances but instead filters out a large number of obstacles from the visibility graph  $VG$  that do not intersect the current shortest path between a data point  $p$  and query points in  $Q$ . Thus, MPAOD may retrieve the

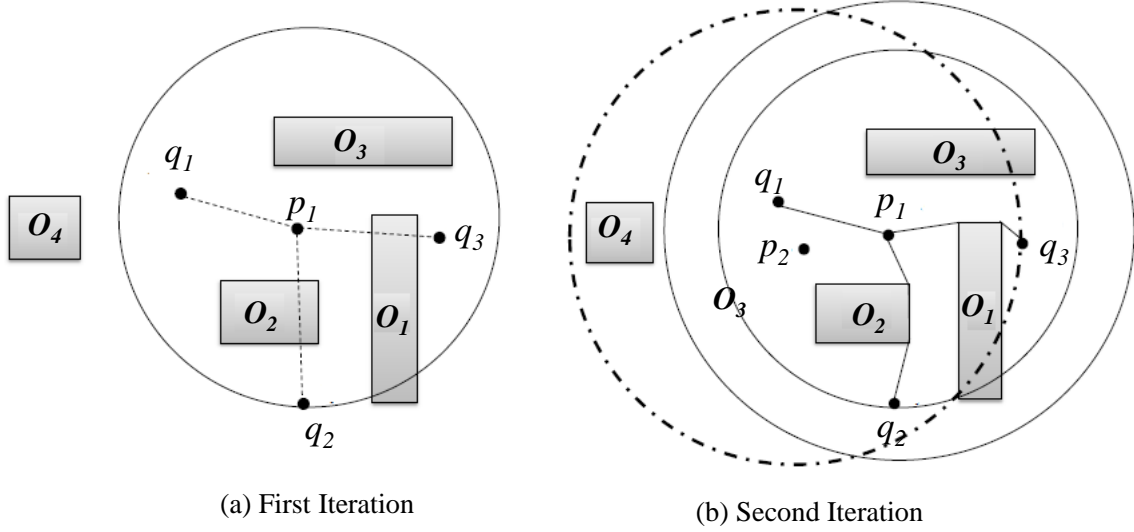


Figure 2.2: Multi Point Based Aggregate Obstructed Distance (MPAOD)

same obstacle multiple times but keeps the visibility graph small. MPAOD works better in a distribution, where the data points are located far apart from each other and the probability of retrieving the same obstacle multiple times is small.

From Figure 2.2, we see that there are three query points,  $Q = \{q_1, q_2, q_3\}$  and a data point  $p_1$ . We want to calculate the aggregate obstructed distance between  $p_1$  and  $Q$ . MPAOD at first compute individual Euclidean distance from each query point  $q_i$  to a data point  $p_1$  and assigns them as initial obstructed distances. Then it incrementally retrieves all obstacles  $o_1, o_2$  and  $o_3$ , that are within the distance,  $dmax = \max_{i=1}^3 dist_O(p_1, q_i)$  centering the data point  $p_1$ . Since  $o_3$  does not intersect with the any of the shortest path, MPAOD rejects it. After filtering out unnecessary obstacles MPAOD updates the visibility graph with the new obstacles,  $o_1, o_2$  and recomputes the obstructed distance of  $q_2$  and  $q_3$ . Then again the algorithm checks for incremental retrieval of obstacles within the range  $\max_{i=1}^3 dist_O(p_1, q_i)$ . Since no obstacle affect the shortest path, the algorithm stops retrieving obstacles and applying the aggregate function SUM or MAX, we get the aggregate obstructed distance  $dist_{AO}(p_1, Q)$  from  $p_1$  to  $Q$ . For a new data point  $p_2$ , MPAOD retrieves 4 obstacles  $o_1, o_2, o_3$  and  $o_4$ , among which only  $o_4$  is new and  $o_1, o_2, o_3$  were retrieved by MPAOD for  $p_1$ .

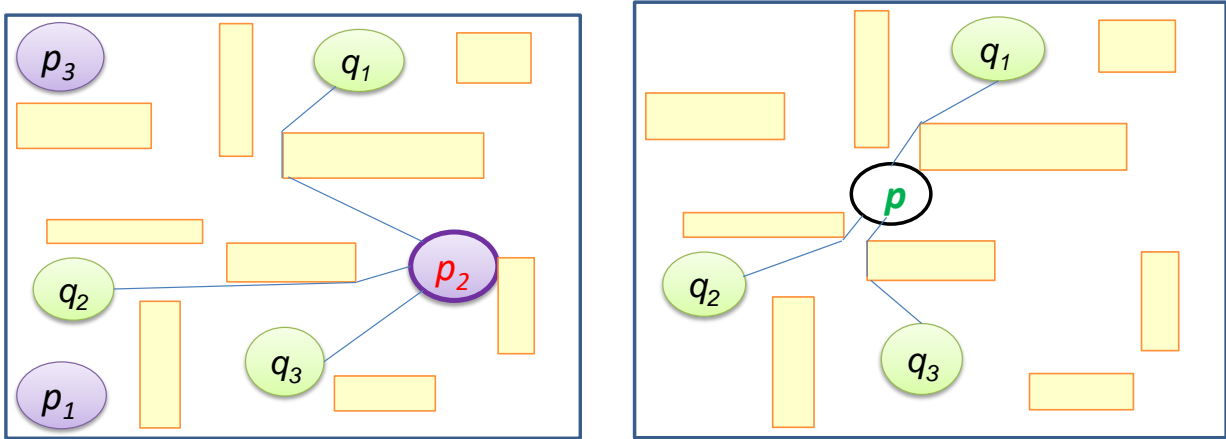


Figure 2.3: OGNN query vs. OOMP query

### 2.4.5.2 OGNN query vs. OOMP query

OGNN queries are very much related to OMP queries in obstructed space. The key difference between the OGNN query and OOMP query is, An OGNN query, returns the location of a POI from the given set of POIs, that minimizes the total obstructed travel distance with respect to the locations of the group members, whereas in case of an OOMP query a meeting point does not need to be at the location of a POI, it can be anywhere in the obstructed space except the areas of obstacles. Therefore, the OOMP query is more difficult than the OGNN query due to its infinite search space.

For example, in the Figure 2.3, an OGNN query, returns the location of a POI (Point Of Interest),  $p_2$  from the given set of 3 POIs ( $p_1, p_2, p_3$ ), that minimizes the total obstructed travel distance with respect to the location of three group members ( $q_1, q_2, q_3$ ). Whereas in case of an OOMP query a meeting point  $p$ , does not need to be at the location of a POI, it can be anywhere in the obstructed space except the areas of obstacles. Therefore, the OOMP query is more difficult than the OGNN query due to its infinite search space.

### 2.4.6 Visible $k$ Nearest Neighbor ( $k$ VNN) Queries

In the obstructed space, another class of queries are visible  $k$  nearest neighbor queries. Two points  $p$  and  $q$  are mutually visible to each other, if the open line segment joining them does not intersect the interior of any obstacle. Visible  $k$  nearest neighbor queries returns  $k$  nearest data points which are not blocked by any obstacles. In [15] and [39], continuous visible  $k$  nearest neighbor queries ( $k$ CVNN)



are discussed and algorithms are proposed, which are based on R-tree based indexing. In [14] and [40] Gao and Zheng proposed reverse visible  $k$  nearest neighbor queries, which efficiently reduces the preprocessing time and prune the search space through half-plane property and visibility check. All visible  $k$  nearest neighbor queries are proposed in [41], which retrieves visible  $k$  nearest neighbors for each point in a query set  $Q$ .

### 2.4.7 Group Visible Nearest Neighbor (GVNN) Queries

In [13] a new type of query is proposed called a Group Visible Nearest Neighbor Query (GVNN), which prunes both data set and obstacle set by defining the invisible region of *minimum bounded rectangle* (MBR) of query set. This is the first research which relates obstacles with GNN queries. The paper presents two algorithms Multiple Traversing Obstacles (MTO) algorithm and Traversing Obstacles Once (TOO) algorithm to efficiently solve GVNN problem. GVNN queries has some difference with our OOMP queries. GVNN queries prune those data points which cannot be seen by all the query points. Consider a set of query points  $Q$  for which we are going to answer GVNN query, if there is a data point  $p$  which is blocked by an obstacle, the GVNN query rejects  $p$  though it is visible from all the other query points in  $Q$ . On the other hand, our OOMP query considers the total obstructed distance between the candidate point location and the query point set  $Q$  and does not prune the invisible points in the obstructed space.

## 2.5 Indexing Techniques for Space Partitioning

A number of indexing techniques for space partitioning are available such as, Kd-trees, Quadtrees, Octrees, R-trees [42–46] etc. R-tree has number of benefits over other indexing techniques. That's why on spatial database, researchers use R-tree and its variants [47–49] most frequently. R-tree is a depth-balanced tree where each node corresponds to a minimum bounding rectangle. Each leaf node consists of an array of leaf entries and each non-leaf node contains an array of node entries. An example of R-tree is showed in Figure 2.4 which indexes a set of query points.

Here we discuss the differences among R-tree, kd-tree, octree and quad-tree in brief. For example, kd-trees [42, 43] partition the whole space into regions whereas R-trees only partition the subset of space containing the points of interest. Kd-trees represent a disjoint partition (points belong to only one

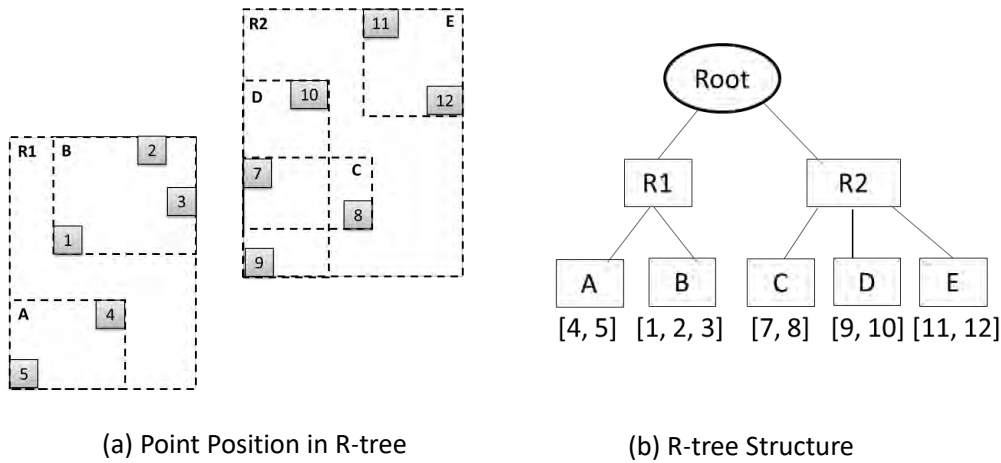


Figure 2.4: Example of R-tree

region) whereas the regions in an R-tree may overlap. R-trees are disk-oriented whereas kd-trees are memory oriented. The most important benefit of R-tree is that, it can store rectangles and polygons but kd-trees can only store point vectors. So, R-trees are much more suitable in an obstructed space than kd-trees. On the other hand, quadtrees [44, 45] are most often used to partition a two dimensional space by recursively subdividing it into four quadrants or regions whereas R-tree is a balanced search tree and it organizes the data in pages, and is designed for storage on disk. Quadtrees work only on two dimensional space but R-trees also work in multi dimensional space. R-trees requires less storage than quadtrees. Octrees [46] are the three-dimensional analog of quadtrees. For nearest neighbor queries, R-tree indexes are faster than octrees and quadtrees. We have used R-trees as the indexing structure for our obstacles, though the algorithms are applicable for any indexing technique.

### 2.5.1 R-tree Indexing Structure

R-tree performs the traversing of the tree in a branch-and-bound manner. For nearest neighbor search algorithms, two types of technique exists, the depth-first and the best-first.

First of all, the depth-first algorithm starts traversing from the root of the tree and visits the node whose minimum distance (MinDist) from the query object is smallest. In this way, when it reaches a leaf node, it finds the candidate nearest neighbor. After that, it traverse back and only visits the nodes whose minimum distance is smaller than the distance of the candidate nearest neighbor.

The best-first nearest neighbor search is known as the incremental nearest neighbor search. It maintains a priority queue for the visited nodes. The queue stores entries in an increasing order of their minimum distance from the query object. The entry with the smallest minimum distance is visited first. The best-first nearest neighbor search gives the advantage of retrieving the successive nearest neighbors in an incremental manner without re-computing the query from the scratch. We have used the best-first nearest neighbor search approach in this thesis to retrieve entries from R-tree.

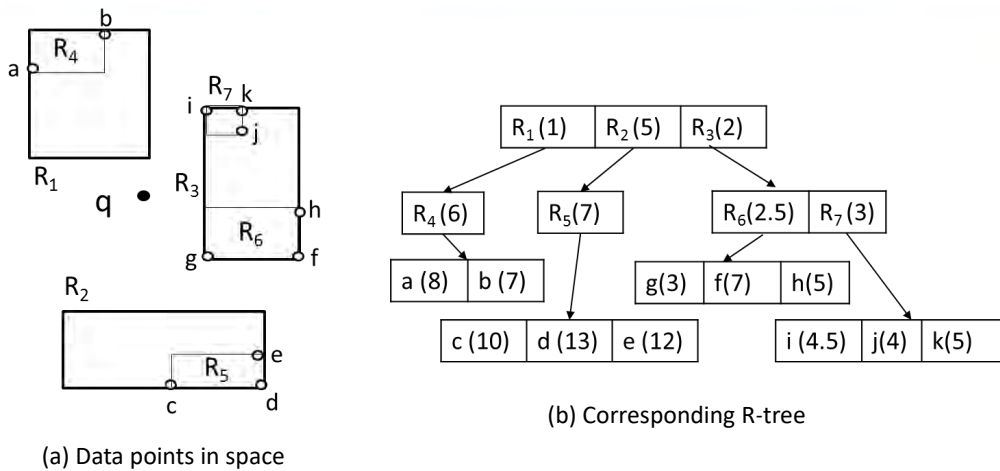


Figure 2.5: A best first search example in R-tree

Figure 2.5 shows an example of best first search (BFS) in R-tree. Figure 2.5a shows the data points  $a, b, c, d, e, f, g, h, i, j, k$  in this space and Figure 2.5b shows the corresponding R-tree. The steps to retrieve the nearest neighbor of query point  $q$  is shown in Table 2.1, where the first column shows the action in R-tree and the rest of the columns show the entries in the priority queue. The first nearest neighbor  $g$  of query point  $q$  is reported at the 6<sup>th</sup> iteration.

Table 2.1: BFS in R-tree

Visit Root	$R_1$	$R_3$	$R_2$					
Follow $R_1$	$R_3$	$R_2$	$R_4$					
Follow $R_3$	$R_6$	$R_7$	$R_2$	$R_4$				
Follow $R_6$	$R_7$	$g$	$R_2$	$h$	$R_4$	$f$		
Follow $R_7$	$g$	$j$	$i$	$R_2$	$h$	$k$	$R_4$	$f$
Report $g$	$j$	$i$	$R_2$	$h$	$k$	$R_4$	$f$	

The researchers find the optimal meeting point for a given set of query points in an Euclidean space and road network space. But, none of them considered obstacles in the path. In this thesis we propose

---

the novel approach to find the OOMP which includes relevant obstacles and search space in a branch and bound manner by pruning out irrelevant obstacles and search space. The irrelevant obstacles are those obstacles which does intersect the shortest path between any two points in the obstructed space.

## Chapter 3

# Obstructed Distance Computation

In this chapter, we propose our algorithms for minimum and maximum total obstructed distance computation. In Section 3.1, we describe Visibility Graph, which is used throughout our algorithms. In Section 3.2, we discuss existing obstructed distance computation algorithms between two points. In Section 2.4.5.1 we describe the algorithms for computing aggregate obstructed distance computation strategies. In Section 3.3 we give the details of the algorithms for computing minimum and maximum total obstructed distance from a square quadrant to all query points.

### 3.1 Visibility Graph

To compute obstructed distance among the points we need to build visibility graph. Visibility graph consists of all the obstacle vertices along with the query points. There is an edge between two nodes in the visibility graph if and only if the two nodes are mutually visible to each other i.e. no obstacle edge obstructing the visibility between them [50].

Figure 3.1 shows an example of visibility graph. There are three obstacles,  $O_1$ ,  $O_2$  and  $O_3$  and two points  $p$  and  $q$ . All the visible vertices are joined by edges and the shortest obstructed path between  $p$  and  $q$  are showed by a green solid line. An important property of visibility graph is, in an obstructed space the edges of the shortest path are the edges of the visibility graph [51].

A number of algorithms [52–54] have been developed for the construction of visibility graph. John et al. [55] compared the developed algorithms and showed each of the algorithm’s benefits and limitations. The naive algorithm requires  $O(n^3)$  time. The first nontrivial solution to the visibility

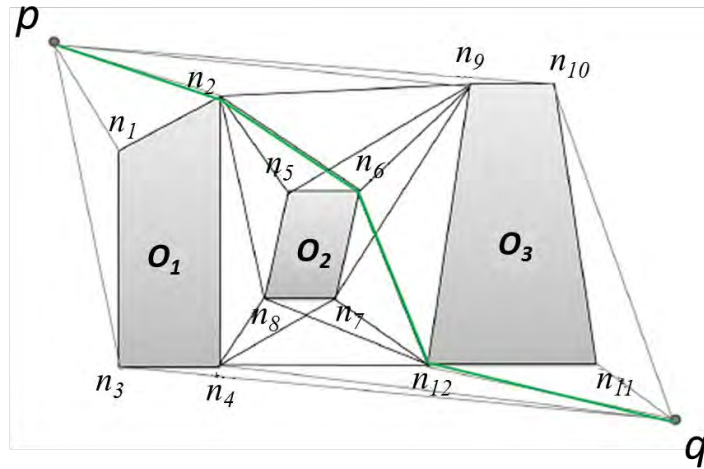


Figure 3.1: Obstructed distance computation through visibility graph

problem is given by Lee [52]. The running time of the algorithm is  $O(n^2 \log n)$ . The other two algorithms [53, 54] are based on complex data structure which are good for theoretical interest. In this thesis, we have used the algorithm proposed by Lee et al. [52] for constructing visibility graph.

In our research problem, the size of spatial dataset is huge. It is not a good choice to keep the whole visibility graph in the main memory. That's why, we construct the visibility graph incrementally. We add only those obstacles and data points in the graph which are relevant to the query. Whenever in the graph any new obstacle or data point is added, we efficiently update the graph. We also remove obstacles which are not relevant for our query and keep the size of the visible graph small.

### 3.1.1 Incremental Visibility Graph Computation Strategy

To update an existing visibility graph, the authors in [34] introduced the following strategies which we have also used for the **incremental visibility graph construction** :

**Adding a new obstacle** : To add a new obstacle  $o$  in the visibility graph  $G$ , we add all the vertices  $V$  of  $o$  to the graph and create new edges with all the visible vertices from  $V$ . If there are any existing edges that crosses the interior of  $o$ , we remove them from the graph.

**Adding a new query point** : To add a new query point  $q$  in the visibility graph  $G$ , we add  $q$  in

the graph and find all the visible vertices  $V$  of  $q$  and create new edges between them.

**Deleting an existing query point :** To remove a query point  $q$  from the visibility graph  $G$ , we just simply remove the point and its incident edges from the graph.

### 3.2 Obstructed Distance between two Points

A number of algorithms have been developed [34–36] for computing obstructed distance between two points. Now we give a brief overview of the algorithms with their limitations. First attempt of computing obstructed distance between two points was taken by Zhang [34] and later improved in [35]. We describe the algorithm proposed by Zhang using the example shown in Figure 3.2.

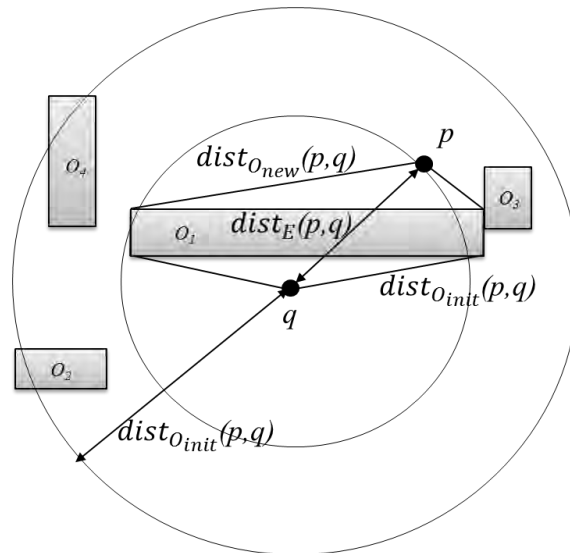


Figure 3.2: Obstructed distance computation between  $p$  and  $q$

Initially the algorithm computes Euclidean distance  $dist_E(p, q)$  between two points. Then the algorithm retrieves the obstacles within the circle centered at  $q$  and the radius of the circle is  $dist_E(p, q)$ . The retrieved obstacle is  $o_1$ . Initial visibility graph is constructed with  $o_1, p$  and  $q$ . As the obstacle  $o_1$  intersects the Euclidean straight line between  $p$  and  $q$ , obstructed distance between  $p$  and  $q$  needs to be computed. Using Visibility graph and Dijkstra shortest path algorithm, initial obstructed distance  $dist_{O_{init}}(p, q)$  is computed. But  $dist_{O_{init}}(p, q)$  might not be real obstructed distance, as other obstacles

lying outside of the current circular area might intersect, which may affect the actual obstructed distance.

Thus the algorithm again executes a circular range query, where the center of the circle is  $p$  and the radius is  $dist_{O_{init}}(p, q)$ . The new set of retrieved obstacles are  $o_2$ ,  $o_3$  and  $o_4$ . Thus we update the visibility graph by adding these new three obstacles and recompute the actual obstructed distance,  $dist_{O_{new}}(p, q)$ . The algorithm repeats the circular range query iteratively until no new obstacle is added in the visibility graph which affect the obstructed distance between  $p$  and  $q$ .

Second algorithm proposed by Xia [35] introduced the idea of retrieving obstacles incrementally which filters out a large number of obstacles. Thus the size of visibility graph is kept small. Initially the visibility graph contains the two points  $p$  and  $q$  and the obstacle which intersects the straight line distance  $dist_E(p, q)$  between the two points. Then the algorithm computes the obstructed distance  $dist_O(p, q)$  and check whether any obstacle which is not included in the current visibility graph, intersects the distance,  $dist_O(p, q)$ . If any obstacle intersects then, it is included in the visibility graph and the obstructed distance is recomputed again. The algorithm stops checking when no more obstacle further intersects the obstructed shortest path.

Gao proposed an Algorithm [10] which introduces the idea of reusing the already computed obstructed distance between the points and other obstacle vertices. They expand the visibility graph incrementally and maintain a threshold. They proved that, if the obstructed distance  $dist_O(p, q)$  between a data point  $p$  and a query point  $q$  is computed by considering all the obstacles in the region bounded by a threshold from the query point  $q$ , and  $dist_O(p, q) \leq \text{threshold}$  and then  $dist_O(p, q)$  is the real obstructed distance between the data point  $p$  and the query point  $q$ .

These obstructed distance calculation strategies do not describe how to compute obstructed distance from a single data point to multiple query points. By applying the already existing algorithms recursively between the data point and each of the query points  $q \in Q$  incurs high query processing overhead. In the next section, we discuss the aggregate obstructed distance computation strategies.



### 3.3 Minimum and Maximum Total Obstructed Distance Computation

In this section, we propose algorithms for computing minimum and maximum total obstructed distance from a square to query points. In the Euclidean space there is algorithm for computing minimum and maximum distance from a point to rectangle. However, when obstacles are present, then the maximum and minimum euclidian distance might not be the real minimum and maximum obstructed distance. In the obstructed space no algorithm has been yet developed for computing minimum obstructed distance from a point to a square. In [11] Gao developed algorithm for computing maximum obstructed distance from a point to a query line segment. We have applied their idea to compute maximum total obstructed distance from query points to a square. Now our proposed algorithms are discussed in the following subsections.

Table 3.1 summarizes the notations used in the rest of this chapter.

Table 3.1: Notations and their meanings

Symbol	Meaning
$q$	A user's location (a query point)
$Q$	A set of query points $\{q_1, q_2, \dots, q_n\}$
$O$	A set of obstacles $\{o_1, o_2, \dots, o_n\}$
$R$	The square quadrant
$R_{center}$	center point of the square quadrant
$ip$	intersection point lies on the boundary line of the square
$RT_O$	an obstacle R-tree
$LVG$	a local visibility graph
$distMin_O(q, R)$	minimum obstructed distance from query point to a Square
$distMin_E(q, R)$	minimum Euclidean distance from query point to a Square
$distMax_O(q, R)$	maximum obstructed distance from query point to a Square
$SP(v, R)$	Euclidean shortest path from obstacle vertex $v$ to $R$
$dist_O(q, v)$	obstructed distance between $q$ and $v$
$MinDist_{TO}(Q, R)$	minimum total obstructed distance between $Q$ and $R$
$MaxDist_{TO}(Q, R)$	maximum total obstructed distance between $Q$ and $R$

### 3.3.1 Minimum Obstructed Distance Computation from a Point to a Square

Computing minimum obstructed distance from a point to a square is an exhaustive search, as there are infinite number of points exist on the four boundary lines of the square. The key idea of the algorithm is to prune the points those cannot provide minimum obstructed distance. Algorithm 1 shows the steps of computing minimum obstructed distance from a query point to a square quadrant.

---

**Algorithm 1:** CompMinObsDist( $q, R, LVG$ )

---

**Input:** A query point  $q$ , a quadrant  $R$ , a local visibility graph  $LVG$

**Output:** The minimum obstructed distance  $distMin_O(q, R)$  of  $q$  from  $R$

```

1  $distMin_O \leftarrow \infty$ 
2 foreach  $v \in LVG$  do
3    $dist_O(q, v) \leftarrow computeObsDistance(q, v)$ 
4 while there exists a node in LVG that has not been visited do
5   let  $v \in LVG$  be the one with the smallest obstructed distance from  $q$ 
6   among those nodes not yet visited
7   if  $dist_O(q, v) > distMin_O(q, R)$  then
8     break
9    $distMin_E(v, R) \leftarrow computeMinimumEuclideanDistance(v, R)$ 
10  if no obstacle intersects SP( $v, R$ ) then
11     $newDistMin_O(q, R) \leftarrow dist_O(q, v) + distMin_E(v, R)$ 
12    if  $newDistMin_O(q, R) < distMin_O(q, R)$  then
13       $distMin_O(q, R) \leftarrow newDistMin_O(q, R)$ 
14 return  $distMin_O(q, R)$ 

```

---

Algorithm 1 at first calls Dijkstras algorithm for computing obstructed distance from a query point  $q$  to all the obstacle vertices in  $LVG$  (Line 2-3). In every iteration, the algorithm consider the vertices in ascending order of obstructed distances from  $q$  (Line 5-6). The algorithm terminates whenever any of the obstacle vertex's obstructed distance,  $dist_O(q, v)$  is greater than already computed minimum obstructed distance,  $distMin_O(q, R)$  (Line 7-8). Otherwise the minimum Euclidean distance between a quadrant and obstacle vertex,  $distMin_E(v, R)$  is calculated (Line-9). Then the algorithm checks whether any obstacle from  $LVG$  intersects this Euclidean shortest path from  $q$  to  $R$ ,  $SP_{(q,R)}$  (Line-10). If any obstacle intersects the shortest path, then the algorithm goes to next iteration and consider the next obstacle vertex. Otherwise, the new minimum obstructed distance  $newDistMin_O(q, R)$  through vertex  $v$  is computed (Line-11), which is the sum of  $dist_O(q, v)$  and  $distMin_E(v, R)$ . Now if this new distance is less than the previous minimum obstructed distance,  $distMin_O(q, R)$  an update is

performed (Line-12-13). Otherwise, the algorithm goes to next iteration without any update and consider the next obstacle vertex. When all the obstacle vertices are visited, the algorithm terminates and returns the minimum obstructed distance  $distMin_O(q, R)$  between a query point and a quadrant (Line 10).

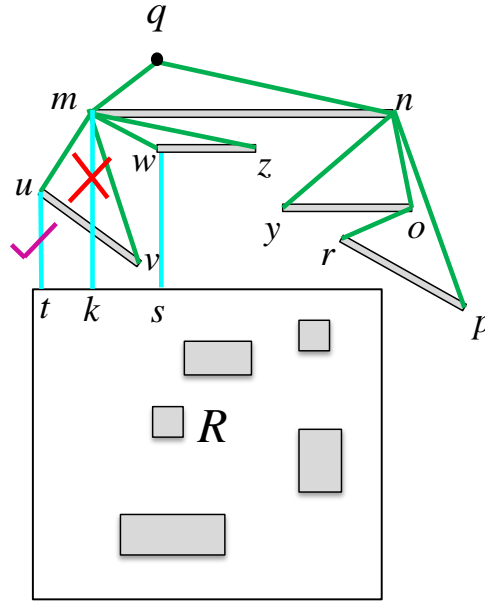


Figure 3.3: Minimum obstructed distance computation example

Consider the following example of Figure 3.3. We will compute minimum obstructed distance from a query point  $q$  to a square quadrant  $R$ . The Figure shows the first three iterations of computation. Here the algorithm at first consider the nearest obstacle vertex  $m$  from  $q$ . But the shortest path  $SP(m, R)$  is intersected by obstacles. Hence the algorithm consider next obstacle vertex  $w$  in ascending order of obstructed distance from  $q$ . Since no obstacle intersects the shortest path from  $w$  to  $R$ , the algorithm computes the minimum obstructed distance from  $q$ ,  $newDistMin_O(q, R)$  through  $w$  which is the sum of  $distMin_O(q, w)$  and  $distMin_E(w, R)$ . The next obstacle vertex is  $u$ , where the new minimum obstructed distance,  $newDistMin_O(q, R)$  through  $u$  is smaller than the previous minimum obstructed distance,  $distMin_O(q, R)$  through  $w$ . Hence update is performed. Now the current updated minimum obstructed path from  $q$  to  $R$  is:  $q \rightarrow m \rightarrow u \rightarrow t$ . The other vertices i.e.  $z, v, n, o, y$  are considered respectively in ascending order of obstructed distance from  $q$  but no update is performed. The algorithm stops comparing the remaining obstacle vertices i.e.  $r, p$ . Since

the obstructed distance of  $y$  from  $q$ ,  $dist_O(q, y)$  is greater than current minimum obstructed distance  $distMin_O(q, R)$ , the remaining vertices  $(r, p)$  will never optimize the shortest path. Thus the real minimum obstructed distance is the sum of  $dist_O(q, u)$  and  $distMin_E(u, R)$  and the shortest path is  $: q \rightarrow m \rightarrow u \rightarrow t$ .

The following lemma illustrates the correctness of the algorithm:

**Lemma 3.3.1.** *Algorithm  $CompMinObsDist$  returns the minimum obstructed distance from a query point  $q \in Q$  to a quadrant  $R$ .*

**Proof** (*By contradiction*). Let  $v'$  be a one of the obstacle vertex in  $LVG$  witch minimizes the minimum obstructed distance from a query point,  $q$  to a quadrant,  $R$ . The vertex  $v'$  may not be considered in the shortest path of minimum obstructed distance for two reasons: (i) The vertex  $v'$  has not been considered as the Euclidean distance from  $v'$  to  $R$  intersects the obstacles. (ii) the algorithm has been terminated before considering the vertex  $v'$ .

The algorithm consider the obstacle vertices in ascending order of obstructed distance  $dist_O(q, v)$  from  $q$ . The minimum obstructed distance from a  $q$  to a quadrant  $R$  is  $distMin_O(q, R) = dist_O(q, v) + distMin_E(v, R)$ . Hence, the algorithm computes  $distMin_E(v, R)$  and checks whether any obstacle intersects the  $SP(v, R)$ . However, if any obstacle intersects the shortest path  $SP(v, R)$  then the real obstructed distance from  $v$  to  $R$  go through others vertices i.e.  $v_1, v_2, v_m$ , which are the corner vertices of obstacles not considered yet. Thus the algorithm goes to the next iteration and consider another vertex. Hence,  $v'$  is only ignored, if any obstacle intersects the straight line distance  $distMin_E(v', R)$  between  $v'$  and  $R$ .

Further, the algorithm terminates only when any vertex,  $v$  is found, whose obstructed distance  $dist_O(q, v)$  is greater than the current value of minimum obstructed distance  $distMin_O(q, R)$ . Since the vertices from the sorted list are considered in ascending order of  $dist_O(q, v)$  in the algorithm, the remaining vertices would never optimize the shortest path. Thus if  $v'$  is not considered in the computation of shortest path of minimum obstructed distance, then according to our algorithm either the minimum Euclidean distance from  $v'$  to  $R$  is intersected by any obstacle or  $v'$  has obstructed distance which is larger than the current value of minimum obstructed distance  $distMin_O(q, R)$ , which again contradicts the assumption. Therefore, our assumption is invalid and the proof completes.

□

### 3.3.2 Minimum Total Obstructed Distance Computation from Multiple Points to a Square

Now we can apply our developed Algorithm 1 for computing minimum obstructed distance from each of the query points to a square and finally summing up the individual minimum obstructed distances we get the minimum total obstructed distance from all query points to a square. However, this algorithm would incur extremely high processing overhead, as same obstacle is retrieved multiple times from the database to compute multiple individual obstructed distances and finding the obstructed distance between two locations is an expensive computation. Thus, we develop Algorithm 2 for computing the minimum total obstructed distance from multiple query points  $Q$  to a square  $R$ . The key idea of this algorithm is, when computing minimum total obstructed distance between a square  $R$  and a set of query points  $Q$ , we retrieve obstacles having distances equal to a threshold with respect to the center point of the square  $R$ . This technique reduces the number of obstacles retrieved from the database, and does not retrieve the same obstacle multiple times from the database. The algorithm also checks the intersection of the retrieved obstacles with the latest shortest path. Thus prunes a huge number of obstacles and keeps the visibility graph small. A small visibility graph makes the obstructed distance computations faster.

Algorithm 2 shows the steps of computing minimum total obstructed distance,  $MinDist_{TO}(Q, R)$  from a quadrant,  $R$  to all query points,  $Q$ . The algorithm at first computes individual Euclidean distances between the quadrant  $R$  and each of the query points  $q \in Q$  and assigns them as the initial minimum obstructed distances between  $R$  and  $q \in Q$ , respectively (Lines 1-2). Then individual Euclidean distance is computed from the center of the quadrant  $R_{center}$  to intersection point  $ip$  which lies on the boundary line of the square quadrant  $R$ . The algorithm finds the distance computed in this step as  $d_{max}$  (Line 4). Then it incrementally retrieves all obstacles (except the obstacles inside the quadrant) that are within the distance  $d_{max}$ , centering the quadrant  $R_{center}$  by using a function  $IOR(R_{center}, RT_O, d_{max})$  (Line 5). The intuition behind retrieving obstacles centering the quadrant is, it is expected that the obstacles that really effects the minimum obstructed distances between  $q \in Q$  and  $R$  is located near  $R$ . An obstacle which is far away from  $R$  cannot actually affect the minimum obstructed distance between the query point  $q \in Q$  and the quadrant  $R$ .

**Algorithm 2:**  $\text{CompMinTotalObsDist}(Q, R, RT_O, LVG)$ 


---

**Input:** A set of query points  $Q = \{q_1, q_2, \dots, q_n\}$ , a quadrant  $R$ , an obstacle R-tree  $RT_O$ , a local visibility graph  $LVG$

**Output:** The minimum total obstructed distance  $\text{MinDist}_{TO}(Q, R)$  of  $Q$  from  $R$

```

1 foreach  $q \in Q$  do
2    $\lfloor \text{distMin}_O(q_i, R) = \text{distMin}_E(q_i, R)$ 
3 repeat
4    $d_{max} \leftarrow \max_{i=1}^n (\text{distMin}_O(q_i, R) + \text{dist}_E(ip, R_{center}))$ 
5    $O \leftarrow \text{IOR}(R, RT_O, d_{max})$ 
6   foreach  $o \in O$  do
7     foreach  $q \in Q$  do
8       if  $o$  intersects  $SP_{q,R}$  then
9         Add  $q$  in  $L_Q$ 
10        Add  $o$  in  $LVG$ 
11   foreach  $q \in L_Q$  do
12      $\lfloor \text{distMin}_O(q, R) = \text{compMinObsDist}(q, R, LVG)$ 
13 until  $L_Q = \emptyset$ 
14  $\text{MinDist}_{TO}(Q, R) \leftarrow \sum_{i=1}^n \text{distMin}_O(q_i, R)$ 
15 return  $\text{MinDist}_{TO}$ 

```

---

However, after retrieving the obstacles, the algorithm filters out those obstacles which do not intersect any of the already computed minimum obstructed distance between the quadrant  $R$  and the query points  $Q$ . It also stores query points in a set  $L_Q$ , whose minimum obstructed distances need to be recomputed. We denote the minimum obstructed distance between a quadrant  $R$  and a query point  $q$  as  $\text{distMin}_O(q, R)$  (Lines 6-10). The recomputation of the minimum obstructed distance,  $\text{distMin}_O(q, R)$  between a quadrant  $R$  and a query point  $q$  is required only when the minimum obstructed distance between  $R$  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 minimum obstructed distance between  $R$  and all the query points  $q \in L_Q$  using the the function  $\text{compMinObsDist}(q, R, LVG)$  (Lines 7-12). The procedure repeats until the minimum obstructed distance intersects no new obstacles or  $L_Q$  is empty (Lines 11-13).

Finally, the algorithm calculates the minimum total obstructed distances,  $\text{MinDist}_{TO}(Q, R)$  by computing the summation of  $\text{distMin}_O(q, R)$  of each query point  $q \in Q$  (Lines 14-15) and returns the

minimum total obstructed distance  $MinDist_{TO}(Q, R)$ .

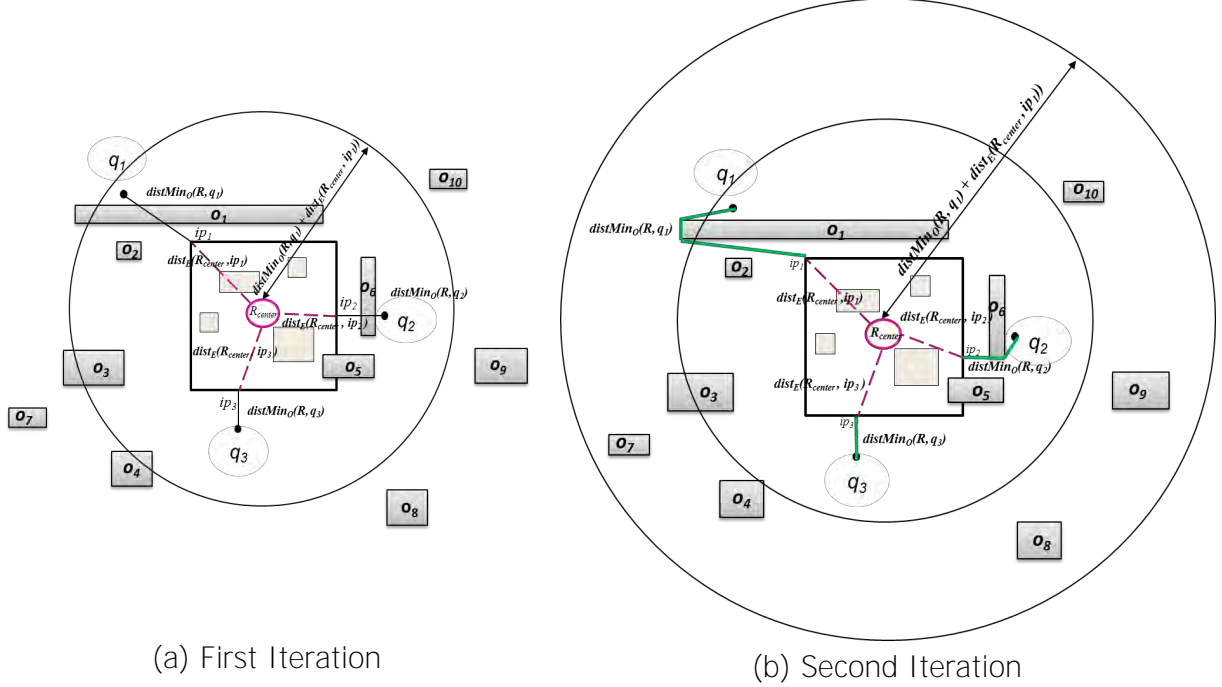


Figure 3.4: Minimum total obstructed distance computation

Consider the following Figure as an example. There are three query points  $q_1$ ,  $q_2$  and  $q_3$  and the quadrant  $R$ . Figure 3.4(a) shows the first iteration of Algorithm 2. In the first iteration, three Euclidean straight line from each query point to the quadrant is drawn which intersects at three points  $ip_1$ ,  $ip_2$  and  $ip_3$  respectively (shown in the solid line) and these distances are defined as  $distMin_O(q_1, R) = dist_E(q_1, ip_1)$ ,  $distMin_O(q_2, R) = dist_E(q_2, ip_2)$  and  $distMin_O(q_3, R) = dist_E(q_3, ip_3)$ . Then individual Euclidean distance is computed from the center of the quadrant  $R_{center}$  to the intersection points  $ip_1$ ,  $ip_2$  and  $ip_3$  respectively, which lies on the boundary line of the square quadrant  $R$  (shown in the dotted line). The algorithm finds the distance  $d_{max}$  (Line 4) as  $distMin_O(q_1, R) + dist_E(ip_1, R_{center})$ . Then it incrementally retrieves all obstacles (except the obstacles inside the quadrant) that are within the distance  $d_{max}$ , centering the quadrant  $R_{center}$  by using a function  $IOR(R_{center}, RT_O, d_{max})$  (Line 5). The incremental obstacle retrieval algorithm retrieves obstacle  $o_1, o_2, o_3, o_4$  and  $o_5$ . The Euclidean shortest path between the query point  $q_1$  and the quadrant  $R$ , intersects obstacle  $o_1$ , and the Euclidean shortest path between the query point  $q_2$  and the quadrant  $R$ , intersects obstacle  $o_6$ . Thus  $q_1$  and  $q_2$  are inserted in  $L_Q$  and obstacles  $o_1$  and  $o_6$  are added to  $LVG$ . The minimum obstructed

distance from  $q_1$  to  $R$  and from  $q_2$  to  $R$  is updated.

Figure 3.4(b) shows the second iteration of Algorithm 2, where more obstacles ( $o_7, o_8, o_9, o_{10}$ ) are retrieved by the incremental obstacle retrieval and checked for intersection with the query points  $Q$ . Since, the new shortest paths do not intersect any of the obstacles, so the algorithm terminates after computing the minimum total obstructed distance.

### 3.3.2.1 Correctness of Algorithm *CompMinTotalObsDist*

The following lemma shows the correctness of the algorithm.

**Lemma 3.3.2.** *Algorithm *CompMinTotalObsDist* finds the minimum total obstructed distance from a quadrant  $R$  to a set of query points  $Q = (q_1, q_2, \dots, q_n)$ .*

**Proof** (*By contradiction*). Let, there be an obstacle  $o$ , which is not retrieved *CompMinTotalObsDist* (Algorithm 2) and  $o$  changes the minimum total obstructed distance  $distMin_{TO}(R, Q)$  computed by algorithm *CompMinTotalObsDist*. Algorithm *CompMinTotalObsDist* incrementally retrieves obstacles that block current minimum obstructed distance from  $R$  to all the query points in  $Q$ . If there are no new obstacles are retrieved that blocks the current minimum obstructed distance from  $R$  to all the query points in  $Q$ , the algorithm computes the minimum total obstructed distance  $distMin_{TO}(R, Q)$  between  $R$  and  $Q$ .

In [35], Xia proves that, if an obstacle does not intersect the current shortest path between two points in the visibility graph, then that obstacle cannot change the already computed obstructed distance between those points.

Since  $o$  is not retrieved by Algorithm *CompMinTotalObsDist* so it does not intersect any of the shortest path from  $R$  to the query points in  $Q$ . Here shortest path for each query point, is the minimum obstructed distance between two points ( $q$  and any point lies on the boundary of quadrant  $R$ ). Thus, the obstacle  $o$  cannot change any of the minimum obstructed distances between  $R$  and a set of query points  $Q$ . Since, minimum total obstructed distance  $distMin_{TO}(Q, R)$  is computed by summing up all the individual minimum obstructed distances  $distMin_O(q, R)$ ,  $o$  cannot change total minimum obstructed distance  $distMin_{TO}(Q, R)$  too, which contradicts our assumption. Thus algorithm



*CompMinTotalObsDist* computes the actual minimum total obstructed distance  $distMin_{TO}(Q, R)$  between a quadrant  $R$  and a set of query points  $Q$  and the proof completes.

□

### 3.3.2.2 Time Complexity of Algorithm CompMinTotalObsDist

Let,  $|Q|$  is the number of query points,  $|V|$  is the number of nodes in the visibility graph,  $|E|$  is the number of edges in the visibility graph,  $|Ob|$  is the number of obstacles retrieved from the R-tree.

The **for** loop in lines 2-3, takes  $O(|E| + |V|\log|V|)$  times to compute obstructed distance [34] from the query point to all the nodes in *LVG*. After then all the nodes are sorted in ascending order of obstructed distance from the query point which takes  $O(|V|\log|V|)$  time. The algorithm in lines 7-8, takes constant time to check whether the  $dist_O(q, v)$  is less than  $distMin_O(q, R)$ . Computation of minimum Euclidean distance takes constant time (In line 9). In order to check whether any obstacle (from the  $|Ob|$  obstacles) intersects the Euclidean shortest path from  $v$  to  $R$  takes  $O(|Ob|)$  time (Line-10). The remaining operations (line 11-13) take constant amount of time. In the worst case, the body of the while loop takes  $O(|V|)$  times since there are  $|V|$  number of nodes (assumed) in the visibility graph.

Thus the worst case time complexity of the *CompMinObsDist* algorithm is:

$$O(|E| + |V|\log|V| + |V|\log|V| + |V|^2) = O(|E| + 2|V|\log|V| + |V|^2)$$

In *CompMinTotalObsDist* algorithm, the **foreach** loop in lines 1-2 takes  $|Q|$  times to compute minimum Euclidean distance from each query point to quadrant  $R$ . Inside the repeat loop,  $d_{max}$  is computed in constant time. In line-5, the incremental obstacle retrieval *IOR* from the R-tree,  $RT_O$  takes  $O(|V|^2 \times \log|V|)$  [34] time. The cost of checking whether the obstacle  $o$  intersects the  $SP_{q,R}$  takes constant time. If intersected, then corresponding  $q$  is added to  $L_Q$  in constant time and the cost of adding obstacle in *LVG* takes  $|V|\log|V|$  time [34]. Hence the two nested loop in lines 6-10 takes  $O(|Ob| \times |Q| \times (|V|\log|V|))$  time. The for loop in lines 11-12 takes  $O(|Q|) \times O(|E| + 2|V|\log|V| + |V|^2)$  time for minimum obstructed distance computation from all the query points in the worst case.

Consequently, the worst case time complexity of the *CompMinTotalObsDist* algorithm is:  $O(|Ob| \times ((|V|^2 \times \log|V|) + (|Ob| \times |Q| \times (|V|\log|V|)) + |Q| \times (|E| + 2|V|\log|V| + |V|^2)))$

$$\approx O(|Ob|^2 \times |Q| \times |V| \times \log|V|)$$

### 3.3.3 Maximum Total Obstructed Distance Computation

Algorithm 3, *CompMaxTotalObsDist* finds the maximum total obstructed distance from a square quadrant  $R$  to a set of query points  $Q = (q_1, q_2, \dots, q_n)$ . A quadrant ( $R$ ) is a square, hence there are 4 boundary lines/sides. In [11], Gao et al. gave algorithm for finding the maximum obstructed distance from a point to a line segment. Here in this Algorithm 3, in every iteration, the function *IOR* proposed in [11], incrementally retrieves the obstacles those affect the computation of maximum obstructed distance from a query point to a line (Line-3). After then for each side of the quadrant, maximum obstructed distance  $distMax_O(q, line_k)$  from a query point to a side or line is computed using the function *compMaxObsDist* which calls the control point list computation (*CPLC*) algorithm proposed in [11] (Line-4). Then the largest of these 4 distances is taken as the maximum obstructed distance  $distMax_O(R, q)$  from a query point to a quadrant (Line-5). After summing up the individual maximum obstructed distances of each query point, we get the maximum total obstructed distance  $distMax_{TO}(R, Q)$  from a quadrant to all the query points,  $Q$  (Line-6).

---

**Algorithm 3:** *CompMaxTotalObsDist*( $Q, R, RT_O, LVG$ )

---

**Input:** A set of query points  $Q = \{q_1, q_2, \dots, q_n\}$ , a quadrant  $R$ , an obstacle R-tree  $RT_O$ , a local visibility graph  $LVG$

**Output:** The maximum total obstructed distance  $MaxDist_{TO}(Q, R)$  of  $Q$  from  $R$

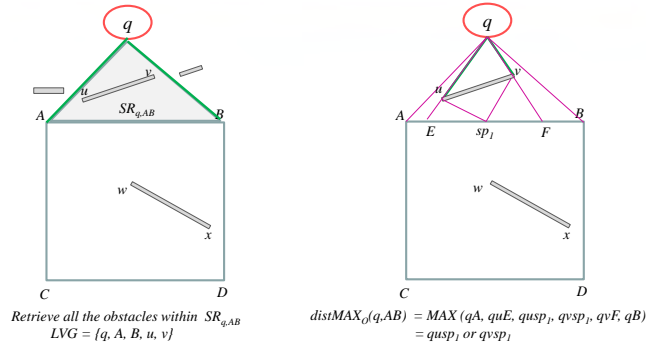
```

1 foreach  $q \in Q$  do
2   foreach  $line_k \in R$  do
3      $LVG \leftarrow IOR(q, RT_O, line_k)$ 
4      $distMax_O(q, line_k) = compMaxObsDist(q, line_k, LVG)$ 
5    $distMax_O(R, q) = \max_{k=1}^4 distMax_O(q, line_k)$ 
6  $MaxDist_{TO}(Q, R) \leftarrow \sum_{i=1}^n distMax_O(R, q_i)$ 
7 return  $MaxDist_{TO}$ 

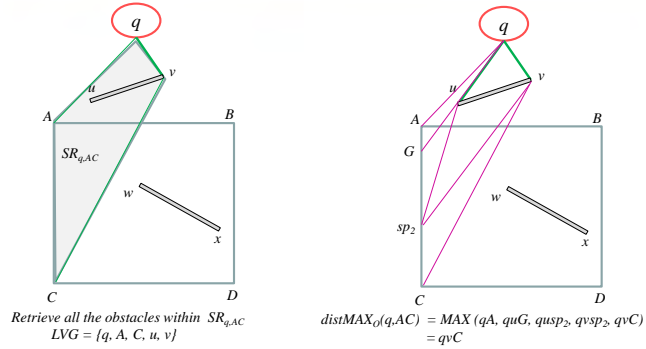
```

---

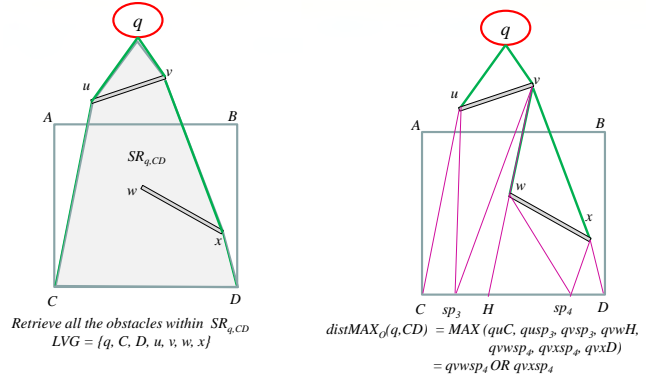
Figure 3.5 shows an example of calculating maximum total obstructed distance computation from a set of query points  $Q$  to a square quadrant  $ABDC$ . Figure 3.5(a), 3.5(b), 3.5(c) and 3.5(d) show the computational steps of maximum obstructed distance from  $q$  to a line segments  $AB, AC, CD$  and  $BD$  respectively.



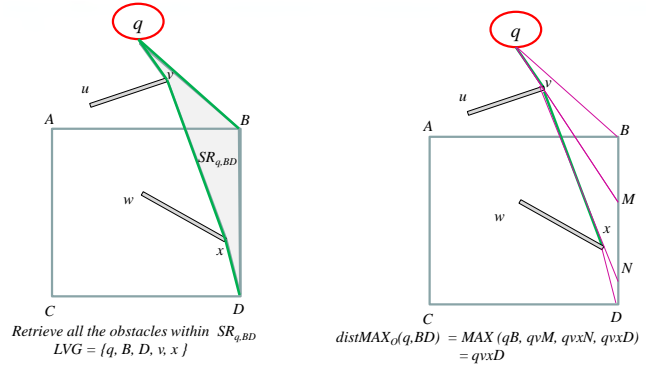
(a) From a query point to AB



(b) From a query point to AC

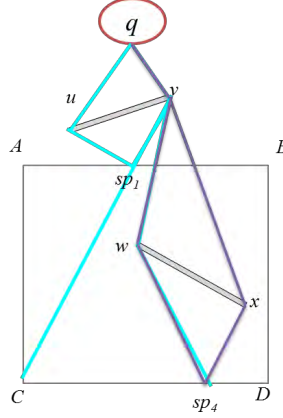


(c) From a query point to CD



(d) From a query point to BD

Figure 3.5: Computation of maximum obstructed distance from a query point to 4 Lines



$$\begin{aligned} \text{distMAX}_O(q, ABDC) &= \text{MAX}(\text{distMAX}_O(q, AB), \text{distMAX}_O(q, AC), \text{distMAX}_O(q, CD), \text{distMAX}_O(q, BD)) \\ &= \text{MAX}(\{qusp_1 \text{ OR } qvsp_1\}, qvC, \{qv wsp_4 \text{ OR } qvxsp_4\}, qvxD) \\ \text{distMAX}_O(q, ABDC) &= qv wsp_4 \text{ OR } qvxsp_4 \end{aligned}$$

Figure 3.6: Computation of maximum obstructed distance from a quadrant to a query point

The incremental obstacle retrieval algorithm, *IOR* at first incrementally retrieves the obstacles within the shaded area of  $SR_{q,AB}$  which is shown in the first Figure of 3.7(a). Since it is proved [11], that any obstacle outside the range of  $SR_{q,AB}$  could not affect the obstructed distances from  $q$  to all the points lying in the straight line  $AB$ . Now we have to find out the point on  $AB$  which is the farthest point from  $q$  considering obstacles. A naive algorithm is to compute obstructed distance of all the points along  $AB$  from  $q$  and take their MAX. However, there are infinite number of points exist along the line segment  $AB$ . So the naive algorithm is not feasible.

In the paper [11], the authors primary focus is on developing algorithms for finding continuous obstructed nearest neighbor *CONN* query that returns nearest neighbors (among the data points or objects lying in the obstructed space) of all the points in a given query line (the trajectory segment through which a client is moving ) in a two-dimensional (2D) space. A naive algorithm is to perform ONN retrieval [34] at every single point of a specified query line segment. However, it is not feasible since there exist unlimited number of points along the line segment. They showed that nearby points along the line segment normally share the same ONN. They decompose the given line segment into a number of smaller segments/intervals with each having its own ONN. Thus, it is only necessary to issue ONN search at those points where ONN objects change. Hence, the concept of split point is introduced in [56]. The *split point* can be defined as:

**Definition 3.3.1.** Let the two end points of the line segment,  $l$  are start and end.  $O$  represents the

obstacle set and  $p_1$  and  $p_2$  are the data points. Then the line can be split into two smaller segments,  $[start, sp]$  and  $[sp, end]$ , where  $p_1$  is the ONN to all the points along  $[start, sp]$  and  $p_2$  be the ONN for all the points along  $[sp, end]$ , point  $sp$  is a split point where the ONN corresponding to  $l$  changes.

In order to facilitate the formation of split points the authors introduce a novel concept, namely *control point* which can be defined as:

**Definition 3.3.2.** Given  $p$ : a data point,  $O$ :an obstacle set, and an interval  $R$  over the given line segment, a point  $cp$  is the control point of  $p$  over  $R$ , iff (i) the shortest path from  $p$  to any point on  $R$  passes through  $cp$ ; and (ii)  $cp$  is visible to every point on  $R$ .

Consider the following example of second Figure 3.5(a). Here the line segment  $AB$  is decomposed into four intervals,  $[A, E]$ ,  $[E, sp_1]$ ,  $[sp_1, F]$  and  $[F, B]$ . The split points are  $E$ ,  $sp_1$  and  $F$  and  $u$  and  $v$  are the control points are over the interval  $[E, sp_1]$  and  $[sp_1, F]$  respectively. Point  $u$  is the control point for the query point  $q$  over the line segment  $[E, sp_1]$ , meaning that for any point,  $p$  which lies on the line segment  $[E, sp_1]$ , the shortest path from  $q$  to  $p$  must pass  $u$ , and the obstructed distance between  $q$  and  $p$  is the length of the shortest obstacle-free path,  $|SP(q, p)|$  which is the sum of  $|SP(q, u)|$  and Euclidean distance,  $dist_E(u, p)$ .

In [11], the authors utilized a quadratic function to compute the split points, and exploit quadratic characteristics to quickly determine the intervals (bounded by the split points). Moreover, several pruning strategies and optimizations are proposed to further improve the search performance. The Control Point List Computation (*CPLC*) Algorithm in [11] computes obstructed distance from  $q$  to all the split points and end points of the intervals. Taking the maximum distance of these distances give the maximum obstructed distance from  $q$  to  $AB$ .

Hence the maximum obstructed distance is computed using the following formula [11]:

Let the control point list of  $q$  over the line segment  $AB$ , i.e.,  $CPL_{q,AB} = \{cp_i, R_i\}$  with  $i \in [1, m]$ , let  $distMax_O = MAX_{i \in [1, m]} (|SP(q, cp_i)| + dist_E(cp_i, R_i.l), |SP(q, cp_i)| + dist_E(cp_i, R_i.r))$ , where  $m$  is number of line segments/intervals of  $AB$ ,  $cp_i$  is the control point over interval  $R_i = [R_i.l, R_i.r]$ .

In our Algorithm 3 , we repeatedly called the function *compMaxObsDist* which in turn calls the CPLC algorithm [11] for computing maximum obstructed distance from  $q$  to all the four side lines of the square quadrant. Then the maximum obstructed distance from  $q$  to  $ABDC$  square quadrant

is computed as,  $\max_{i=1}^4 dist_O(q, line_i)$ . After summing up the maximum distances for every query point, we get the total maximum obstructed distance,  $\sum_{i=1}^n distMax_O(q_i, ABDC)$ .

### 3.3.3.1 Time Complexity of Algorithm CompMaxTotalObsDist

Let,  $|Q|$  is the number of query points,  $|V|$  is the number of nodes in the visibility graph,  $|T_O|$  is the number of obstacles in the R-tree  $RT_O$  and  $|T|$  is the maximal number of tuples/intervals in the control point list computation.

CompMaxTotalObsDist algorithm takes  $O(\log|T_O| \times |V| \times \log|V|)$  for IOR [11], takes  $O(|V| \times \log|V| \times |T|^2)$  for maximum obstructed distance computation [11]. In every square quadrant,  $R$  there are 4 boundary lines. Consequently, the worst case time complexity of the CompMaxTotalObsDist algorithm is:

$$O(|Q| \times 4 \times ((\log|T_O| \times |V| \times \log|V|) + (|V| \times \log|V| \times |T|^2)))$$

Since,  $|T| \ll |T_O|$ , thus the complexity:

$$\approx O(4|Q| \times \log|T_O| \times |V| \times \log|V|)$$

## Chapter 4

# Our Algorithms

Two heuristic algorithms are developed for processing *OOMP* (*Obstructed Optimal Meeting Point*) queries. We call our first heuristic algorithm as the hierarchical algorithm because it refines the search space in a recursive manner. On the other hand, we call our second heuristic algorithm as the grid algorithm as it divides the refined search space into a grid. The key difference between the two algorithms are: the hierarchical algorithm hierarchically refines the search space by exploiting geometric properties until obstructed optimal meeting point is identified. It gives an approximate meeting point of pedestrians with accuracy guarantee. Here query processing overhead increases with the increase of group size and obstacles. On the other hand, the grid algorithm converts the initial search space into a number of binary grid cells, which are candidates for optimal meeting points. Here expensive obstructed distance computation is avoided, thus accuracy is sacrificed more compared to hierarchical algorithm but incurs less query processing overhead. In Sections 4.1 and 4.2, we discuss the proposed algorithms which finds OOMP that minimizes the total obstructed distance of all the query points.

Table 4.1 summarizes the notations used in the rest of this chapter.

### 4.1 Hierarchical Algorithm

The key idea of our hierarchical algorithm is to search the obstructed space for an *OOMP* in a hierarchical manner. The search starts by considering the total space. In every iteration of the search, the search space is divided into four quadrants and the quadrant that has the highest probability to include the *OOMP* is considered in the next iteration. The quadrants that cannot have the *OOMP*

Table 4.1: Notations and their meanings

Symbol	Meaning
$q_i$	A user's location (a query point)
$Q$	A set of query points $\{q_1, q_2, \dots, q_n\}$
$O$	A set of obstacles $\{o_1, o_2, \dots, o_n\}$
$R_j$	The $j^{th}$ square quadrant, where $j \in 1,2,3,4$
$R_{sq}$	Squared bounded region
$R$	Dequeued squared quadrant from $list_Q$ which has the smallest minimum total obstructed distance among the quadrants already enqueued
$list_Q$	List of quadrants sorted in ascending order of minimum total obstructed distance from all query points
$MinDist$	The smallest maximum total obstructed distance among all the quadrants visited so far
$Dist_{TO}(Q, R_{jcenter})$	Total obstructed distance between quadrant center point, $R_{jcenter}$ and all the query points, $Q$
$distMin_{TO}(Q, R_j)$	Minimum total obstructed distance between quadrant $R_j$ and $Q$
$distMax_{TO}(Q, R_j)$	Maximum total obstructed distance between quadrant $R_j$ and $Q$
$v$	A vertex of $R_{sq}$ (a square grid cell)
$sl$	Side Length/Square Unit of a grid cell
$dist(v, q_i)$	The shortest distance between $v$ and $q_i$
$dist_{sum}(v, Q)$	The total shortest distance between $v$ and $Q$
$dist_{oomp}$	obstructed optimal meeting point distance



are gradually pruned by exploiting geometric properties.

The algorithm stops splitting the quadrants when the selected quadrant satisfies the user defined threshold. Then the center point,  $R_{center}$  is enqueued as the representative point of the quadrant with the total obstructed distance from  $R_{center}$  to query points. Here a threshold is considered as the side length/square unit of the quadrant. The search terminates when a point is dequeued from the priority queue and it is returned as an *OOMP* with particular accuracy level or approximation ratio. The approximation ratio is defined as the ratio of total obstructed distance from  $R_{center}$  to  $Q$  and the minimum total obstructed distance,  $distMin_{TO}$  from the quadrant (which at first satisfies the threshold) to query points. For example, if the approximation ratio is 1.02, then it is guaranteed that the optimal total obstructed distance for a group lies within 100 to 102. Thus the reported meeting point has maximum 2% error. When the approximation ratio is 1, then the algorithm returns the optimal answer. The more smaller is the threshold, the more accurate the meeting point would be.

Algorithm 4 shows the pseudocode of the hierarchical algorithm for finding optimal meeting point which minimizes the total obstructed distance of the pedestrians. The inputs of the algorithm are the location of  $n$  pedestrians,  $Q = \{q_1, q_2, \dots, q_n\}$  and an obstacle R-tree,  $RT_O$  and a threshold  $sl$ . The algorithm returns the obstructed optimal meeting point *oomp* with approximation ratio as output.

The algorithm works in two steps. The first step is to bound the initial search space. Algorithm 5 shows the steps of bounding region. As in *OOMP* query, no point of interest is given, so the search space is infinite. In [18] Sultana proves that, the obstructed group nearest neighbor for a set of query points  $Q$  lies within the radius  $r$  centered at the median of  $Q$ ,  $c_Q$ , where,  $r$  is the average of total obstructed distance,  $avg\_dist_{TO}(c_Q, Q)$  for the Euclidean nearest neighbor of centroid  $c_Q$ . Any point outside the distance  $r$  from  $c_Q$  gives higher total obstructed distance than all the point inside  $r$ .

Since any point outside the circle with radius  $avg\_dist_{TO}(c_Q, Q)$ , has higher total obstructed distance, we consider only the points inside this circle as candidates for optimal meeting points. But in our problem of *OOMP* query, no point of interest (*POI*) is given. We can choose any point in the obstructed space. But we select Euclidean geometric centroid  $c_Q$  as first candidate for *OOMP*, because centroid has the probability of smallest total obstructed distance from  $Q$  than any other arbitrary

**Algorithm 4:** HierarchicalAlgorithm( $Q, RT_O$ )

---

**Input:** A set of query points  $Q = \{q_1, q_2, \dots, q_n\}$ , an obstacle R-tree  $RT_O$   
 and threshold  $sl$

**Output:**  $oomp$ , an optimal meeting point of  $Q$

```

1   $R_{sq} \leftarrow \text{FindBoundedRegion}(Q, RT_O)$ 
2   $MinDist \leftarrow \infty$ 
3   $distMin_{TO}(Q, R_{sq}) \leftarrow \text{CompMinTotalObsDist}(Q, R_{sq})$ 
4   $distMax_{TO}(Q, R_{sq}) \leftarrow \text{CompMaxTotalObsDist}(Q, R_{sq})$ 
5   $Enqueue(list_Q, R_{sq}, distMin_{TO}(Q, R_{sq}), distMax_{TO}(Q, R_{sq}))$ 
6   $isFirstQuadrant \leftarrow true$ 
7  do
8     $\{ R, distMin_{TO}(Q, R), distMax_{TO}(Q, R) \} \leftarrow \text{Dequeue}(list_Q)$ 
9    if  $R$  is a point then
10      $oomp \leftarrow R$ 
11      $approximation\ ratio \leftarrow dist_{TO}(Q, oomp) / lowerBound$ 
12     return  $oomp$ 
13   else if side length of  $R < sl$  then
14     if  $isFirstQuadrant$  then
15        $lowerBound \leftarrow distMin_{TO}(Q, R)$ 
16        $isFirstQuadrant \leftarrow false$ 
17     end
18      $dist_{TO}(Q, R_{center}) \leftarrow \text{CompAggObsDist}(Q, R_{center})$ 
19     if  $dist_{TO}(Q, R_{center}) \leq MinDist$  then
20        $Enqueue(list_Q, R, dist_{TO}(Q, R_{center}), dist_{TO}(Q, R_{center}))$ 
21        $MinDist \leftarrow dist_{TO}(Q, R_{center})$ 
22     end
23   else
24     Divide  $R$  into four quadrants
25     for each quadrant  $R_j \in R$  do
26        $distMin_{TO}(Q, R_j) \leftarrow \text{CompMinTotalObsDist}(Q, R_j)$ 
27        $distMax_{TO}(Q, R_j) \leftarrow \text{CompMaxTotalObsDist}(Q, R_j)$ 
28       if  $distMin_{TO}(Q, R_j) \leq MinDist$  then
29          $Enqueue(list_Q, R_j, distMin_{TO}(Q, R_j), distMax_{TO}(Q, R_j))$ 
30         if  $distMax_{TO}(Q, R_j) < MinDist$  then
31            $MinDist \leftarrow distMax_{TO}(Q, R_j)$ 
32         end
33       end
34     end
35   end
36 end
37 while ( $list_Q \neq \emptyset$ )

```

---

**Algorithm 5:** FindBoundedRegion( $Q, RT_O$ )

---

**Input:** A set of query points  $Q = \{q_1, q_2, \dots, q_n\}$ , an obstacle R-tree  $RT_O$  and a local visibility graph  $LVG$

**Output:** A bounded region,  $mbsq$  enclosing all candidate points for optimal meeting points of  $Q$

- 1 Initialize( $LVG, c_Q$ )
- 2  $dist_{TO}(c_Q, Q) \leftarrow CompAggObsDist(Q, c_Q, RT_O, LVG)$
- 3  $avg\_dist_{TO} \leftarrow dist_{TO}(c_Q, Q)/|Q|$
- 4  $x_{min} = c_Q - avg\_dist_{TO}, y_{min} = c_Q - avg\_dist_{TO}$
- 5  $x_{max} = c_Q + avg\_dist_{TO}, y_{max} = c_Q + avg\_dist_{TO}$
- 6  $R_{sq} \leftarrow boundSquareRegion(x_{min}, y_{min}, x_{max}, y_{max})$
- 7 **return**  $R_{sq}$

---

points. So, we calculate average of total obstructed distance  $avg\_dist_{TO}(c_Q, Q)$  between centroid and query points and bound the infinite search space using the function  $FindBoundedRegion(Q, RT_O)$  to a circular area, whose center is the Euclidean geometric centroid of the location of the pedestrians and the radius of the circle is the average obstructed distance between center and location of pedestrians. Any location outside the circle can not be an obstructed optimal meeting point.

Thus, we will enclose the infinite search space that is the circular area with a minimum bounding square,  $R_{sq}$  (Line 2). Then the algorithm refines the initial search space recursively until the  $OOMP$ s is identified. Initially, the algorithm computes the minimum total obstructed distance ( $distMin_{TO}(Q, R_{sq})$ ) and maximum total obstructed distance ( $distMax_{TO}(Q, R_{sq})$ ) of the bounded square region  $R_{sq}$  from the query points. Algorithm-1,2,3 shows the steps of computing maximum and minimum total obstructed distance between a quadrant and query points. Then  $R_{sq}$  with  $distMin_{TO}$  and  $distMax_{TO}$  is inserted into a priority queue  $list_Q$ . The elements of  $list_Q$  are ordered in order of  $distMin_{TO}$ .

The two reasons behind computing these two distances are:

- To identify which quadrant we should consider first for searching  $OOMP$
- And also to prune the quadrants where it is guaranteed that optimal meeting point never lies.

In each iteration of the search, the algorithm dequeues quadrant  $R$  from  $list_Q$  that has the smallest  $distMin_{TO}$  among all the quadrants enqueued (Line 9). If the side length,  $sl$  of a quadrant  $R$  is less than or equal to the value of user defined threshold, then the  $distMin_{TO}$  of the quadrant is saved as *lowerBound* of optimal meeting distance with respect to the threshold (Line 14-16) and

the center point of the quadrant  $R_{center}$  is enqueued in the  $list_Q$  after computing total obstructed distance (Line 19-20). Whenever any point is dequeued from  $list_Q$ , then it is returned as  $oomp$  with approximation ratio, which is defined as the ratio of  $dist_{TO}$  and  $distMin_{TO}$  (Line 10-13). Otherwise,  $R$  is divided into four quadrants  $(R_j)$ ,  $1 \leq j \leq 4$  (Line 25). We use  $MinDist$  to prune the quadrant that cannot contain the  $oomp$ . Here,  $MinDist$  represents the smallest among maximum total obstructed distance of the quadrants enqueued so far.  $MinDist$  is initialized to  $\infty$ . After then, for each quadrant  $distMin_{TO}(Q, R_j)$  and  $distMax_{TO}(Q, R_j)$  is computed (Line 27-28). In line 29-32, if  $distMin_{TO}(Q, R_j)$  is less than  $MinDist$  then the quadrant is enqueued in  $list_Q$  otherwise  $R_j$  is pruned and also if  $distMax_{TO}(Q, R_j)$  is less than  $MinDist$ , then  $MinDist$  is updated with the value of  $distMax_{TO}(Q, R_j)$ .

The following lemma illustrates the pruning condition:

**Lemma 4.1.1.** *A quadrant  $R_j$  can be pruned if  $distMin_{TO}(Q, R_j) > MinDist$ .*

**Proof** (*By contradiction*). Let us consider the quadrant  $R_j$  can not be pruned if  $distMin_{TO}(Q, R_j) > MinDist$ , since the quadrant contains actual optimal meeting  $oomp'$ . We know the total obstructed distance of any point which lies inside any quadrant  $R$ , is in between the minimum ( $distMin_{TO}(Q, R)$ ) and maximum ( $distMax_{TO}(Q, R)$ ) total obstructed distance of the quadrant from  $Q$ . Let  $oomp$  be the reported obstructed optimal meeting point. Thus  $dist_{TO}(oomp, Q) > dist_{TO}(oomp', Q)$ . However,  $MinDist$  represents the smallest among the maximum total obstructed distance of the quadrants already enqueued or the total obstructed distance of the center point of the quadrants already dequeued. Thus  $MinDist$  gives an upper bound of the optimal meeting point and  $distMin_{TO}(Q, R)$  gives the lower bound of the optimal meeting point. The value of optimal meeting point must satisfies the following condition,  $distMin_{TO}(Q, R) \leq dist_{TO}(oomp', Q) \leq MinDist$ .

It is already given that a quadrant is only pruned, when  $distMin_{TO}(Q, R_j) > MinDist$ . The point  $p$  lies within the quadrant  $R_j$ , thus  $dist_{TO}(p, Q) \geq distMin_{TO}(Q, R_j)$  and  $dist_{TO}(oomp', Q) \leq MinDist$ . Thus the assumption,  $dist_{TO}(oomp', Q) < dist_{TO}(oomp, Q)$  is contradictory. Therefore, our assumption is not valid. A quadrant  $R_j$  can be pruned if  $distMin_{TO}(Q, R_j) > MinDist$ . Since no obstructed optimal meeting point could lie within the quadrant  $R_j$ . The proof completes.

□

Thus if any quadrant does not satisfy the pruning condition, then the quadrant is enqueued into  $list_Q$  with  $distMin_{TO}(Q, R_j)$ ,  $distMax_{TO}(Q, R_j)$  (Line 20-23) and then check for update of  $MinDist$  to keep track the smallest maximum total obstructed distance of the quadrant already enqueued. Recursively the algorithm continue the process until the locations of  $oomp$  is found (Line 10) or the queue  $list_Q$  is empty (Line 37).

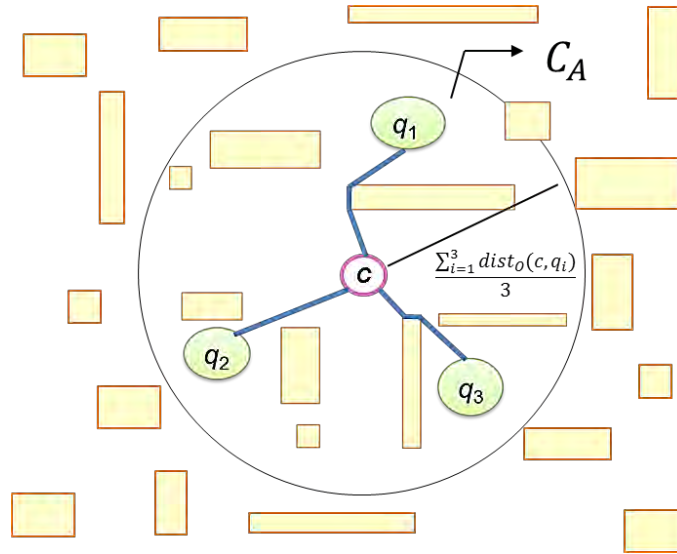


Figure 4.1: Initial bounded search space,  $R_{sq}$  of hierarchical algorithm

For example, consider the following Figure:4.1 and Figure:4.2. Let us consider a scenario, where  $q_1, q_2, q_3$  represents the location of 3 pedestrians and rectangles represent obstacles. Figure:4.1 shows the initial bounded search space, a circular area  $C_A$  whose center  $c$  is the Euclidean geometric centroid of the location of the pedestrians. The radius of the circle is the average distance between center  $c$  and location of pedestrians.

The Figure:4.2 shows the scenario of recursively refining the initial search space. Here, minimum total obstructed distance of quadrant  $R_4$ , is greater than current smallest maximum total obstructed distance of already enqueued quadrants. So we prune quadrant  $R_4$  as  $OOMP$  will never lie here. So  $OOMP$  will lie within quadrants  $R_1, R_2$  and  $R_3$ . Suppose we got the second quadrant  $R_2$  as its minimum total obstructed distance is the smallest. Thus we divide  $R_2$  into 4 quadrants  $R_5, R_6, R_7, R_8$  and repeat the steps again as before. Now we will search the quadrants  $R_1, R_3, R_5, R_6, R_7$  and  $R_8$  for  $OOMP$ . By applying the pruning technique  $R_6, R_7$  are pruned and recursively continue the

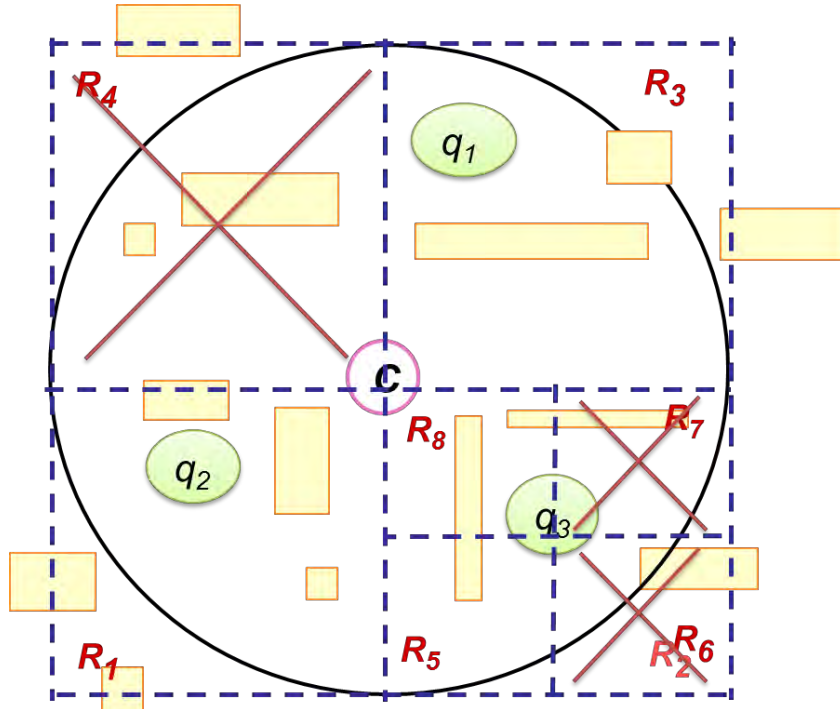


Figure 4.2: Recursively refining the initial search space of hierarchical algorithm

process until we do not find out the location of *OOMP*.

However, the computational time increases rapidly with increase of number of candidate quadrants. Hence we approximate the quadrant (whose side length is less than threshold) with the center point of that square quadrant and the representative point is enqueued for further explore if the total obstructed distance from this representative point to all the query points is less than or equal to *MinDist*, otherwise the point (representing the quadrant) is dropped. Here *MinDist* initially holds the value of smallest maximum total obstructed distance among the squares already enqueued. Later when we get the representative points from the center point of the square (satisfying threshold), then *MinDist* holds the smallest total obstructed distance among the representative points already enqueued.

The algorithm save the the minimum total obstructed distance,  $distMin_{TO}(Q, sq)$  of the first selected quadrant *sq* (whose side length is less than the value of user defined threshold *sl*) dequeued from  $list_Q$  as the *lowerBound*. Since the total distance of the optimal meeting point would not greater than the value of *lowerBound*.

The following lemma illustrates the lower bound condition:

**Lemma 4.1.2.** *The minimum total obstructed distance ( $distMin_{TO}(Q, sq)$ ) of the first selected quadrant (satisfying the threshold),  $sq$  dequeued from priority queue, represents the value of lower bound of the optimal total obstructed distance.*

**Proof** (*By contradiction*). Let us consider, the value of optimal total obstructed distance is greater than the value of the lower bound, which is the minimum total obstructed distance ( $distMin_{TO}(Q, sq)$ ) of the first selected quadrant (satisfying the threshold),  $sq$  dequeued from priority queue.

We know the total obstructed distance of any point which lies inside any quadrant  $R$ , is in between the minimum ( $distMin_{TO}(Q, R)$ ) and maximum ( $distMax_{TO}(Q, R)$ ) total obstructed distance of the quadrant from  $Q$ . In every iteration of the algorithm, the square quadrant is repeatedly divided into four sub-quadrants and each sub-quadrant is enqueued with  $distMin_{TO}$  and  $distMax_{TO}$ , only if  $distMin_{TO}$  is less than or equal to the value of  $MinDist$ , where  $MinDist$  represents the smallest maximum total obstructed distance of the quadrants already enqueued. Thus the optimal meeting point should lie within the quadrants in the queue, whose  $distMin_{TO}$  is less than the value of  $MinDist$  and the optimal meeting distance is in between the smallest  $distMin_{TO}$  and  $MinDist$ . The optimal total obstructed distance  $dist_{TO}(oomp, Q)$  of an optimal meeting point is never less than the value of smallest  $distMin_{TO}$  of the quadrants currently in the queue.

However, the quadrants in the priority queue are sorted in ascending order of  $distMin_{TO}$ . Thus the first quadrant picked from the queue (satisfying the threshold) has the smallest  $distMin_{TO}$  among all the quadrants in the  $list_Q$ . Therefore, our assumption is not valid. The minimum total obstructed distance ( $distMin_{TO}(Q, sq)$ ) of the first selected quadrant (satisfying the threshold),  $sq$  dequeued from priority queue, represents the value of lower bound of the optimal total obstructed distance. The proof completes.

□

Thus the minimum total obstructed distance,  $distMin_{TO}(Q, sq)$  of the first selected quadrant  $sq$  dequeued from  $list_Q$  is considered as the *lowerBound*. Then we take the center point of the quadrant as a representative point and enqueue the center point in the priority queue  $list_Q$ , if the total obstructed distance from this center point to  $Q$  is less than the value of  $MinDist$  and also update the

the value of  $Mindist$ . Also in the next subsequent iterations, we follow the conditions like before. The algorithm terminates when any point has been dequeued from the  $list_Q$ , where elements of  $list_Q$  are maintained in order of  $distMin_{TO}$ . This point's (*oomp*)  $distMin_{TO}$  actually represents the total obstructed distance,  $dist_{TO}(oomp, Q)$  which is less than all the quadrants and points in the queue. Thus *oomp* is returned as approximate meeting point. In order to find out, how much accuracy is lost we measure the approximation ratio.

Let  $Q$  be a set of query points,  $Q = \{q_1, q_2, \dots, q_n\}$ , where  $1 \leq i \leq n$  and  $sq$  is the selected quadrant of side length equal to the user defined threshold  $sl$  which has the smallest minimum total obstructed distance among all the candidate quadrants, then the point *oomp* returned by hierarchical algorithm is the approximate meeting point with approximation ratio  $dist_{TO}(oomp, Q)/distMin_{TO}(sq, Q)$ , where  $dist_{TO}(oomp, Q)$  represents total obstructed distance from *oomp* to  $Q$  and  $distMin_{TO}(sq, Q)$  represents minimum total obstructed distance from the quadrant  $sq$  to  $Q$ .

The derivation of the approximation ratio in terms of problem size and threshold value is out of scope of this research. However, the lower bound mentioned earlier in lemma-4.1.2 has been used to measure the accuracy lost or approximation ratio which is presented in details of the experimental section.

### 4.1.1 Time Complexity of Hierarchical Algorithm

Let,  $|Q|$  is the number of query points,  $|V|$  is the number of nodes in the visibility graph,  $|T_O|$  is the number of obstacles in the R-tree  $RT_O$ ,  $|Ob|$  is the number of obstacles retrieved from the R-tree.

The Hierarchical Algorithm takes  $O(|Ob|^2 \times |Q| \times |V| \log|V|)$  time for `findBoundedRegion`, takes  $O(|V|^2 \times \log|V|)$  for OR (Obstacle Range query)[34] and `CompAggObsDist` algorithm [18] takes  $(|Q| \times |Ob|^2 \times |V| \times \log|V|)$  time. `CompMinTotalObsDist` and `CompMaxTotalObsDist` algorithms take  $O(|Ob|^2 \times |Q| \times |V| \times \log|V|)$  and  $O(4|Q| \times \log|T_O| \times |V| \times \log|V|)$  time respectively. In the for loop of lines 26-35 iterates 4 times which costs  $4 \times (O(|Ob|^2 \times |Q| \times |V| \times \log|V|) + O(4|Q| \times \log|T_O| \times |V| \times \log|V|))$  for computing  $distMin_{TO}$  and  $distMax_{TO}$  of the four square quadrants respectively and then enqueue the values in the  $list_Q$  which takes  $O(1)$  time.



The *Dequeue* function extracts the quadrant with the smallest  $distMin_{TO}$  among the elements already inserted into the priority queue,  $list_Q$ . The cost of dequeue operation depends on the number of quadrants are in the  $list_Q$ . Initially the priority queue contains 4 quadrants. In every iteration the selected quadrant is divided into four sub-quadrants. Let  $N$  be the of number of times the outer do-while loop iterates. Therefore, the length of the priority queue  $list_Q$  is at most  $4 \times i$  in the  $i^{th}$  iteration, where  $1 \leq i \leq N$ . Thus the cost of the *Dequeue* operation is  $O(\log(4 \times i))$ . The inner loop in line 26-35 iterates four times. In the worst case, the outer do-while loop takes  $N \times (\log(4 \times i) + 4 \times (O(|Ob|^2 \times |Q| \times |V| \times \log|V|) + O(4|Q| \times \log|T_O| \times |V| \times \log|V|)))$  time.

Consequently, the worst case time complexity of the hierarchical algorithm is:

$$\begin{aligned}
& O(|Ob|^2 \times |Q| \times |V| \log|V|) + 4 \times ((|Ob|^2 \times |Q| \times |V| \times \log|V|) + (4|Q|^2 \times |Ob| \times |V| \times \log|V|)) + \\
& O(N \times (\log(4 \times i) + 4 \times (O(|Ob|^2 \times |Q| \times |V| \times \log|V|) + O(4|Q| \times \log|T_O| \times |V| \times \log|V|)))) \\
& \approx O(|Ob|^2 \times |Q| \times |V| \log|V|) + O(N \times (\log(4 \times i) + 4 \times (O(|Ob|^2 \times |Q| \times |V| \times \log|V|) + O(4|Q| \times \\
& \log|T_O| \times |V| \times \log|V|)))) \\
& \approx O(|Ob|^2 \times |Q| \times |V| \log|V|) + O(N \times (|Ob|^2 \times |Q| \times |V| \times \log|V|)) \\
& \approx O(N \times (|Ob| \times |Q| \times |V|^2 \times \log|V|)) \text{ [ Since, } |Ob| = 4|V| \text{ ]}
\end{aligned}$$

## 4.2 Grid Algorithm

The query processing overhead increases with the increase of the number of the group members, obstacles and the search space. In practical applications, it is usual to sacrifice the accuracy slightly if the OOMP query can be processed in real time. We bound the infinite search space using minimum bounded square [16, 18]. Then in order to make our algorithm simple, we avoid complex obstructed distance computation techniques [16, 18]. Rather, we convert the search space into a number of binary grid cells, which are candidates of optimal meeting locations. The side length of the grid cell is  $sl$ . The center point of the grid cell is representative point of the corresponding grid cell. The smaller is the number of grid cells, the more accurate our proposed algorithm would be. We applied best first search simultaneously from respective query points grid cell and stops the search until we find the grid cell witch has the smallest total distance among all candidates. Finally the centner point of the

grid cell is returned which is the obstructed meeting point of all query points. The grid algorithm finds the meeting point which incurs reduced processing overhead.

We develop grid algorithm which finds the obstructed meeting locations as well as grid cells of the query points in an obstructed space. Then the center point of the returned obstructed optimal meeting grid cell is considered to be obstructed optimal meeting points of the query points. First of all the initial search space is bounded using the function  $findBoundedRegion()$ . Then it incrementally retrieves all the obstacles that are within the distance  $length_{R_{sq}}/\sqrt{2}$ , centering at the geometric centroid  $c_Q$  of query points,  $Q$  by using a function  $OR(c_Q, RT_O, length_{R_{sq}}/\sqrt{2})$  (Line 2).

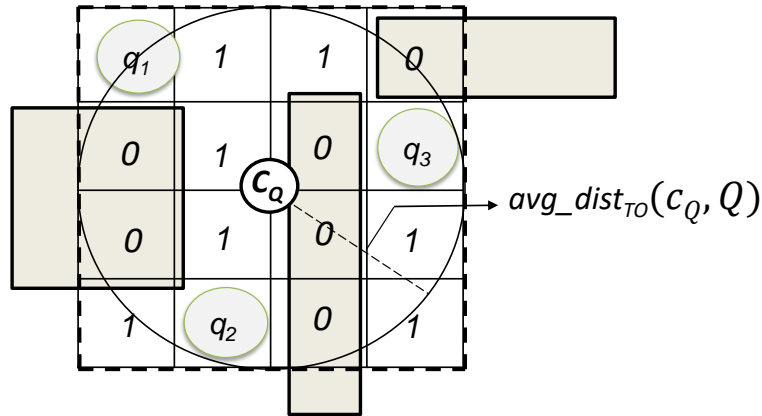


Figure 4.3: Grid representation of bounded square region,  $R_{sq}$

To further reduce the search space, we split the  $R_{sq}$  into a number of grid cells. Thus each grid cell has eight adjacent neighbors. If any obstacle lies in the grid cell, we represent it with zero (where user is not allowed to move), otherwise with one (where user is allowed to move). Now only the square grid cells which are filled with one are candidates for optimal meeting locations. The smaller is the side length,  $sl$  of each grid cell, the larger is the search space but the probability of finding more accurate optimal meeting point is increased. Algorithm 5 shows the pseudocode for finding the bounded square region.

Our grid algorithm is based on best-first search. All the query points or users start best-first search simultaneously from their respective locations. Here grid cells are represented with grid vertices. Separate priority queue is maintained to keep track the list of vertices to be visited from each query

point. The vertices in each priority queue are stored in a order of increasing distance from respective query points. The distance between two adjacent neighbors is the straight line Euclidean distance between the center points of the two neighbors. In every step of the iteration, each query point chooses the most promising vertex form it's queue, which has the smallest distance from respective query point. After dequeuing the vertex, all it's neighbors are enqueued in the priority queue which are not discovered before by  $q_i$ . The search is continued by every query point until all the candidate vertices in the priority queues are visited. Finally the algorithm returns obstructed optimal meeting vertex, which has the smallest total distances from the query points.

Algorithm 6 shows the pseudocode for finding obstructed optimal meeting vertices as well as grid cells. The inputs of the algorithm are set of query points  $Q = \{q_1, q_2, \dots, q_n\}$  and an obstacle R-tree,  $RT_O$ . The algorithm returns the obstructed optimal meeting vertex,  $oomv$  as output.

To bound the infinite obstructed search space, the Algorithm 6 called the function  $findBoundedRegion()$ , which returns square region  $R_{sq}$ . The following lists are used throughout the algorithm:

1.  $dist_{oomv}$ : It is the smallest total shortest distance from the query points to optimal meeting vertex. Initially the distance is initialized with infinity.
2.  $list_Q[1 \dots n]$ : An array of  $n$  lists, where each  $list_Q[i]$  contains a list of vertices from  $R_{sq}$  whose shortest distances from  $q_i$  has already been computed and enqueued. Each  $list_Q[i]$  is sorted in ascending order of shortest distance  $dist(v, q_i)$ , between query point  $q_i$  and grid cell vertex  $v$ .

A function  $Initialize$  is used to initialize each candidate vertex, where  $v = q_i$  as follows:  $\forall_{i=1}^n list_Q[i] \leftarrow q_i$ ,  $\forall_{i=1}^n dist(v, q_i) \leftarrow 0$  and  $count[v] = 0$ .  $count[v]$  is needed to keep track whether each candidate vertex is visited by all query points or not. A  $dist_{oomv}$  is needed to save the distance of obstructed optimal meeting vertex. Initially it is infinite,  $dist_{oomv} \leftarrow \infty$ .

The other functions and symbols that the algorithm uses are listed below:

- $extractMin(list_Q[i])$ : Returns the vertex from the  $i^{th}$   $list_Q$ , whose  $dist(v, q_i)$  is the smallest among all the vertices enqueued in  $list_Q[i]$ .
- $isDiscovered(n_v, q_i)$ : Returns true if a vertex  $n_v$  is seen/discovered by query point  $q_i$ . When a vertex  $n_v$  is seen by  $q_i$ , then  $dist(n_v, q_i)$  has already been calculated and enqueued in  $list_Q[i]$ .

**Algorithm 6:** Grid\_Algorithm ( $Q, RT_O$ )

---

**Input:** A set of query points  $Q = \{q_1, q_2, \dots, q_n\}$ , an obstacle R-tree  $RT_O$ , side length of square grid cell  $sl$

**Output:**  $oomv$ , an obstructed optimal meeting vertex of  $Q$

```

1  $R_{sq} \leftarrow FindboundedRegion(Q, RT_O)$ 
2  $O \leftarrow OR(c_Q, RT_O, length_{R_{sq}}/\sqrt{2})$ 
3 Split  $R_{sq}$  into a number of binary square grid vertices,  $v$ 
4 Initialize ( $list_Q, dist(v, q_i), count, dist_{oomv}$ )
5 repeat
6   foreach  $q_i \in Q$  do
7     if  $list_Q[i] \neq \emptyset$  then
8        $v \leftarrow extractMin(list_Q[i])$ 
9       foreach neighbor of  $v, n_v$  where  $n_v \in R_{sq}$  do
10        if  $(n_v \neq 0)$  and  $(lisDiscovered(n_v, q_i))$  then
11          if  $isDiagonal(n_v, v)$  then
12             $dist(n_v, q_i) \leftarrow dist(v, q_i) + td_1$ 
13          else
14             $dist(n_v, q_i) \leftarrow dist(v, q_i) + td_2$ 
15          end
16          if  $dist(n_v, q_i) \leq dist_{oomv}$  then
17            Add  $n_v$  to  $list_Q[i]$ 
18             $count[n_v]++$ 
19            mark  $n_v$  as seen by  $q_i$ 
20            if  $count[n_v] = |Q|$  then
21               $dist_{sum}(n_v, Q) \leftarrow \sum_{i=1}^n dist(n_v, q_i)$ 
22              if  $dist_{sum}(n_v, Q) < dist_{oomv}$  then
23                update( $dist_{sum}(n_v, Q), dist_{oomv}, oomv$ )
24              end
25            end
26          end
27        end
28      end
29    end
30  end
31 until  $\exists_{i=1}^n list_Q[i] \neq \emptyset$ 
32 return  $oomv$ 

```

---

- $\text{isDiagonal}(n_v, v)$ : Returns true if a vertex  $n_v$  is a diagonal neighbor of vertex  $v$ . In a  $R_{sq}$  a vertex has eight neighbors:  $ul$  (upperLeft),  $um$  (upperMiddle),  $ur$  (upperRight),  $ll$  (lowerLeft),  $lm$  (lowerMiddle),  $lr$  (lowerRight),  $il$  (immediateLeft) and  $ir$  (immediateRight). A vertex is a diagonal neighbor if it is on either  $ul$  or  $ur$  or  $ll$  or  $lr$ .

For each of the query points  $q_i$ , the algorithm extracts the vertex from its corresponding  $i^{\text{th}}$  list (Line 8),  $\text{list}_Q[i]$  whose  $\text{dist}(v, q_i)$  is the smallest among all the vertices enqueued in the  $\text{list}_Q[i]$ .

The algorithm computes shortest distance of all its neighbors from  $q_i$  (Lines 11-14). For each neighbor  $n_v$ , whose value is one (where movement is allowed) and is not yet discovered by query point  $q_i$ , the algorithm computes shortest distance from  $q_i$  to  $n_v$ . Each neighbor's shortest distance,  $\text{dist}(n_v, q_i)$  is the sum of its parent's shortest distance  $\text{dist}_O(v, q_i)$  and the threshold (side length of grid cell,  $sl$ ). For diagonal neighbors the threshold is  $td_1 = (sl \times \sqrt{2})$  (Line-12), otherwise the threshold is  $td_2 = sl$  (Line-14).

The algorithm tries to prune some vertices to reduce candidates from respective  $\text{list}_Q$ . If shortest distance of any vertex  $v$ ,  $\text{dist}(v, q_i)$  is greater than current optimal meeting point distance,  $\text{dist}_{oomv}$ , then the vertex would never become optimal meeting point. So we can prune or cross out this vertex from  $\text{list}_Q[i]$  by  $q_i$ . The algorithm then continue searching for other neighbors.

If pruning condition is not satisfied, then the vertex is added to the  $\text{list}_Q[i]$  for further exploring the search for optimal meeting vertex through  $n_v$  and update the value of counter of the discovered vertex,  $n_v$ . Then mark  $n_v$  as discovered by  $q_i$ . Then the algorithm checks, whether the vertex is discovered by all query points or not (Line 19). If the vertex is discovered by all query points by checking  $\text{count}[v] = |Q|$ , then the algorithm calculates total distance,  $\text{dist}_{sum}(v, Q)$ . If the total distance of this vertex is less than current total distance,  $\text{dist}_{oomv}$  then  $\text{update}()$  function is called to update the optimal meeting vertex. The algorithm repeats the above process for all query points and continues searching for optimal meeting vertex until each list  $\text{list}_Q[i]$  is empty (Line 5-22). At the end the center point of the obstructed optimal meeting vertex is considered as the obstructed optimal meeting point.

Figure 4.4 shows an example of finding obstructed optimal meeting vertex using grid algorithm.

$v_i=q_1$	0	$v_2$	$v_3$
0	$v_4$	0	$v_5$
0	$v_6$	0	$v_7$
0	$v_8=q_2$	0	$v_9=q_3$

Bounded Region:  $R_{sq}$

PRIORITY QUEUE						
step	$list(q_1)$		$list(q_2)$		$list(q_3)$	
1	start from $q_1$	$v_1(0)$	start from $q_2$	$v_8(0)$	start from $q_3$	$v_9(0)$
2	visit $v_1$	$v_4(1.4)$	visit $v_8$	$v_6(1)$	visit $v_9$	$v_7(1)$
3	visit $v_4$	$v_6(2.4), v_2(2.8)$	visit $v_6$	$v_4(2)$	visit $v_7$	$v_5(2)$
4	visit $v_6$	$v_2(2.8), v_8(3.4)$	visit $v_4$	$v_1(3.4), v_2(3.4)$	visit $v_5$	$v_3(3), v_2(3.4)$
5	visit $v_2$	$v_8(3.4), v_3(3.8), v_5(4.2)$	visit $v_1$	$v_2(3.4)$	visit $v_3$	$v_2(3.4)$
6	visit $v_8$	$v_3(3.8), v_5(4.2)$	visit $v_2$	$v_3(4.4), v_5(4.8)$	visit $v_2$	$v_4(4.8)$
7	visit $v_3$	$v_5(4.2)$	visit $v_3$	$v_5(4.8)$	visit $v_4$	$v_6(5.8), v_1(6.2)$
8	visit $v_5$	$v_7(5.2)$	visit $v_5$	$v_7(5.8)$	visit $v_6$	$v_1(6.2), v_8(6.8)$
9	visit $v_7$	$v_9(6.2)$	visit $v_7$	$v_9(6.8)$	visit $v_1$	$v_8(6.8)$
10	visit $v_9$	-----	visit $v_9$	-----	visit $v_8$	-----

step	oomv	$dist_{oomv}$	$dist_{sum}$								
			$v_1$	$v_2$	$v_3$	$v_4$	$v_5$	$v_6$	$v_7$	$v_8$	$v_9$
1	---	---	0	0	0	0	0	0	0	0	0
2	---	---	0	0	0	1.4	0	1	1	0	0
3	---	---	0	2.8	0	3.4	2	3.4	1	0	0
4	$v_2$	9.6	3.4	9.6	3	3.4	6.2	3.4	1	3.4	0
5	$v_2$	9.6	3.4	9.6	6.8	3.4	6.2	3.4	1	3.4	0
6	$v_4$	8.2	3.4	9.6	11.2	8.2	11	3.4	1	3.4	0
7	$v_4$	8.2	9.6	9.6	11.2	8.2	11	9.2	1	3.4	0
8	$v_4$	8.2	9.6	9.6	11.2	8.2	11	9.2	12	10.2	0
9	$v_4$	8.2	9.6	9.6	11.2	8.2	11	9.2	12	10.2	13
10	$v_4$	8.2	9.6	9.6	11.2	8.2	11	9.2	12	10.2	13

Figure 4.4: Best first search for finding optimal meeting vertex of three query points

There are three query points,  $q_1, q_2$  and  $q_3$ . There are 16 vertices. The vertices which are filled with 0 indicating that query user is not allowed to move here. Since these areas are occupied with obstacles. Thus the vertices which are not intersected by obstacles are candidates for optimal meeting points. These candidate vertices are  $v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9$ . In every iteration, the first table shows the vertices enqueued by each query point on the individual queues  $list_Q[i]$  and the distances from the source  $q_i$  to the vertices is also shown in bracket. The second table shows the total distance,  $dist_{SUM}$  of the vertices when discovered by all query users and the update of obstructed optimal meeting vertex,  $oomv$ . After four iterations, vertex  $v_2$  becomes the  $oomv$ . Later the vertex  $v_4$  becomes the  $oomv$ , since  $dist_{SUM}(v_4)$  is less than  $dist_{SUM}(v_2)$ . The algorithm stops searching in a best first manner when all the vertices are visited by all the query points. Finally the center point of the vertex  $v_4$  is returned as  $OOMP$ . It has smallest total distance  $dist_{sum}(v_4, Q) = 1.4 + 2.4 + 4.8 = 8.2$  among all the vertices.

### 4.2.1 Time Complexity of Grid Algorithm

Let,  $|Q|$  is the number of query points,  $|V|$  is the number of nodes in the visibility graph,  $|T_O|$  is the number of obstacles in the R-tree,  $|Ob|$  is the number of obstacles retrieved from the R-tree and  $N$  is the number of vertices in the entire bounded square region. The Grid Algorithm takes  $O(|Q| \times |Ob|^2 \times 4|V| \log|V|)$  time for  $findBoundedRegion()$ , takes  $O(|V|^2 \times \log|V|)$  time for  $OR$  (Obstacle Range query) and  $CompAggObsDist()$  algorithm [18] takes  $(|Ob|^2 \times |Q| \times 4|V| \times \log|V|)$  time.

The repeat loop iterates  $N$  times in the worst case until all the vertices ( $N$ ) are visited by all query points. The inner for loop iterates  $|Q|$  times. In line-8,  $extractMin$  takes  $\log|N|$  time. The foreach loop (line-9) iterates 8 times, since each vertex has at most eight adjacent neighbors. In each iteration, the update operation if satisfied the condition, in line-22, takes constant time.

Thus, the worst case time complexity of the Grid algorithm is:

$$O((|Q| \times |Ob|^2 \times 4|V| \log|V|) + (|V|^2 \times \log|V|) + V + N \times Q \times (\log N + 8 \times N)) \\ \approx O(|Q| \times |Ob|^2 \times 4|V| \times \log|V|)$$

[ Here,  $N = (2r/sl)^2$ , where,  $r = dist_{TO}(c_Q, Q)/|Q|$  ]

## Chapter 5

# Experiments

In this section we present our experimental results of our proposed algorithms. We vary the group size, query distribution area, and the square unit. In each experiment, we evaluate the impact of one parameter while others are fixed at their default values. Table 5.1 summarizes the values for each parameter used in the experiments.

We used real dataset of Germany[57], which have 30674 MBRs of railway lines (rrlines) and 76999 MBRs of hypsography data (hypsogr). We normalized the dataset into an area of  $10,000 \times 10,000$  square unit. From the dataset we used the rrlines as obstacles. Though we considered rectangle shaped obstacles, our algorithm can support any arbitrary shaped polygons. The obstacles are indexed using R-tree in the database and assuming a page size of 4K. We randomly generate the query points which are allowed to lie on the boundary of the obstacles but not inside the obstacles. The query points follow random uniform distribution. The default query area is set to 0.005%, hence the group of query objects are confined to 50% of the entire data space. We performed 100 sample OOMP queries and got the average experimental results.

Parameter	Range	Default
Square Unit	1, 0.5, 0.25, 0.125, 0.0625	1
Group size	2, 4, 8, 16, 32	8
Query rectangle area	0.002% to 0.01%	0.005%

Table 5.1: Experiment setup

We ran our experiments on Intel Core i7-2920XM quad-core CPU (2.50GHz) PC with 32GB RAM.



We measure CPU time and IO cost by varying group size (number of query points), query area (the area to which the group of query objects are confined) and square unit of our proposed algorithms. To evaluate our algorithms for OOMP queries, we measure the approximation error, which gives a measure how much accuracy we have lost. To compute the approximation error, we at first calculate the difference between exact total obstructed distance (upper bound) of the approximate meeting point and the lower bound of the optimal distance and then divide the difference with the lower bound of the optimal distance. The following Equation 5.1 shows the approximation error calculation formula:

$$approximation\_error = \frac{dist_{TO}(oomp, Q) - dist_{MinTO}(sq, Q)}{dist_{MinTO}(sq, Q)} \quad (5.1)$$

For example, if the approximation error is 0.02, then it is guaranteed that the optimal total obstructed distance for a group lies within 100 to 102. Thus the reported meeting point has maximum 2% error. The more closer the value of approximation error is to 0, the more accurate meeting point we will get.

We calculate the exact total obstructed distance of the reported obstructed meeting point, *oomp* and the minimum total obstructed distance from the square area, *sq* (which at first satisfies the given threshold square unit) to all the query points, in grid and hierarchical algorithm of 100 samples by varying square length. Then we calculate error in each sample and then take their average. We evaluate the approximation error by varying the square unit, group size and query area.

## 5.1 Comparison of OOMP Algorithms

We evaluate and compare the Hierarchical and Grid algorithm. We run experiments and all the comparisons are performed in terms of IO cost, computational time and accuracy level.

### 5.1.1 Effect of Group Size

We study the impact of group size on the performance of OOMP query by varying the group size using 2, 4, 8, 16 and 32 and measure the required running time and the number of obstacle access from the obstacle R-tree.

Figure 5.1(a) shows the performance of obstacle R-tree access with the increase of group size. In grid

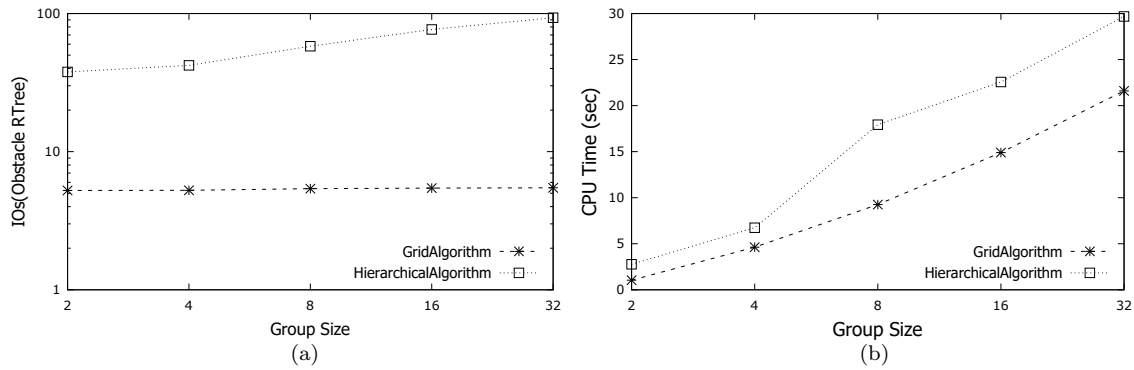


Figure 5.1: Effect of group size

algorithm, we need to access obstacle R-tree initially to bound infinite search region. After computing the total obstructed distance between centroid and query points, we retrieve all the obstacles within the circular area from the geometric centroid and fill the search space with obstacles. That's why IO access almost remain constant on varying group size. However in the hierarchical algorithm, we retrieve obstacles incrementally from the R-tree. Thus, IO access increases rapidly with the increase of group size. We also observe that grid algorithm gives on average 10 times lower IO access than hierarchical algorithm. Since, in the hierarchical algorithm, the search space is refined recursively for computing obstructed distance of the query users, hence IO access increases. Whereas in the grid algorithm the obstacles are retrieved only once while bounding the initial search space. Thus IO access is high in hierarchical algorithm compared to grid algorithm.

Figure 5.1(b) shows the time required by grid and hierarchical algorithm for finding optimal meeting point. It is observed that the performance of both algorithms degrade as the group size increases. For example, for an increase of the group size from 4 to 32, CPU time increases more than 4 times on the average. Since, finding the obstructed distance between two locations is an expensive computation. Thus in the hierarchical algorithm, with the increase of group size increases the number of obstructed distance computations and hence increases more obstacle retrieval from the obstacle R-tree. On the grid algorithm, every grid cell is visited in a best first manner from the location of each pedestrian. Hence the computational time increases with the increase of group size.

Grid algorithm is better in terms of CPU time and IO access than hierarchical algorithm. Hierarchical algorithm gives the meeting location which is very much close to optimal solution by computing

actual obstructed distances among the locations in the bounded search space. Whereas, on the grid algorithm expensive obstructed distance computation is avoided, thus accuracy is sacrificed but incurs less query processing overhead.

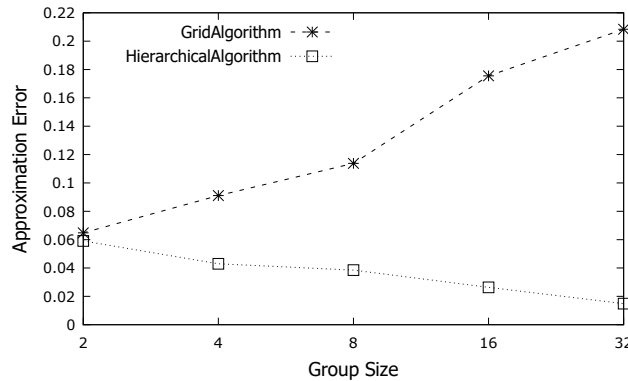


Figure 5.2: Effect of group size on approximation error

We also vary the group size while keeping query area and length of square area at a fixed default value to evaluate the approximation error in both algorithms. Figure 5.2 shows the impact of group size on approximation error. The approximation error decreases with the increase of group size in the hierarchical algorithm, whereas in the grid algorithm the approximation error increases with the increase of group size. Since, in the grid algorithm, the distances are computed based on heuristic function, thus with the increase of group size the accuracy falls. Whereas, in the hierarchical algorithm, the distances are exact obstructed distance, thus accuracy rises with the increase of group size.

### 5.1.2 Effect of Query Rectangle Area

Here we vary the query area, i.e., the area to which the group of query objects are confined to as 0.002%, 0.004%, 0.006%, 0.008% and 0.01% of the entire data space.

We observe that the number of IO access and computation time both increases with the increase of query area and grid algorithm outperforms hierarchical algorithm in terms of IO access and computational time requirements.

Figure 5.3(a) shows that, in grid algorithm number of IO access remains almost constant for different percentage of query area. Whereas, the hierarchical algorithm shows a clear increasing trend with the

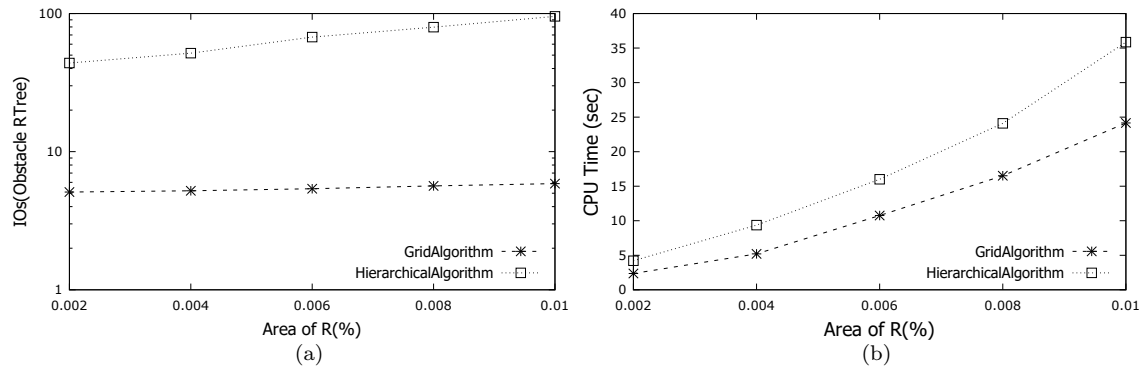


Figure 5.3: Effect of query rectangle area

increase of query area. Figure 5.3(b) shows the curves of computational time in both algorithm. In both scenario, it is observed that the computational time to find the optimal meeting point increases rapidly with the increase of query area. For example, with an increase of query area from 0.004% to .006%, CPU time increases twice.

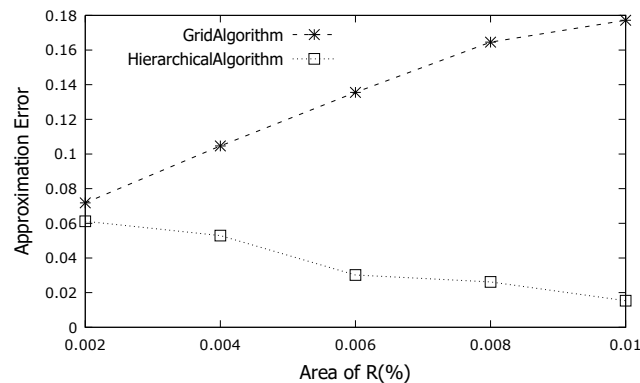


Figure 5.4: Effect of query rectangle area on approximation error

We also vary query area to evaluate the accuracy in both algorithms. Figure 5.4 shows the impact of query area on approximation error. Here, the more far away the users are the more accurate meeting point is found in the hierarchical algorithm. However in the grid algorithm with the increase of query users distribution area, the accuracy falls.

### 5.1.3 Effect of the Length of Square Grid Cell

Here we vary the length of square area, i.e., the area where the optimal meeting point of query objects lies. Here we vary the length of the square area from 1 to .0625 (1, 0.5, 0.25, 0.125, 0.0625).

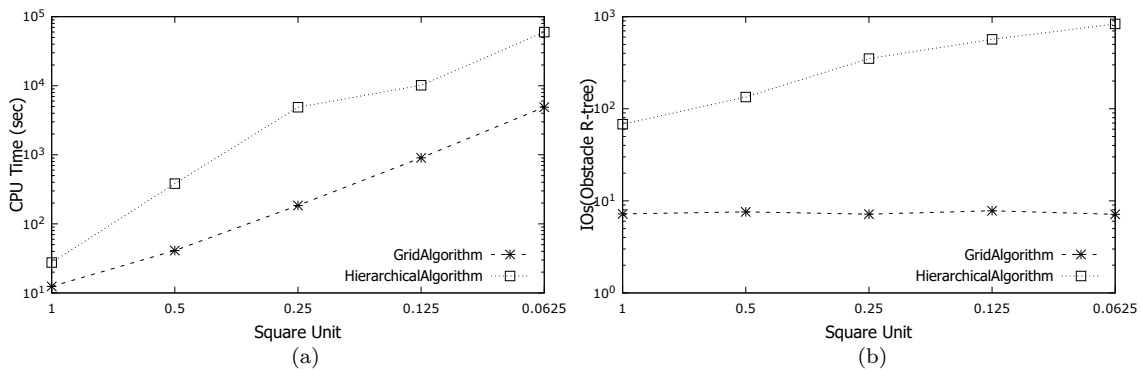


Figure 5.5: Effect of square length

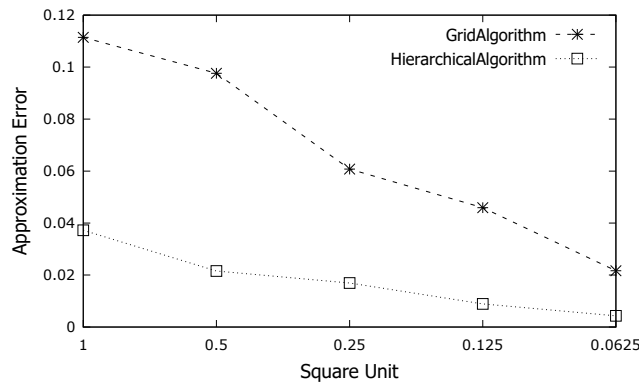


Figure 5.6: Effect of square length on approximation error

Figure 5.5(a) and 5.5(b) shows that, the number of IO access and computation time increases rapidly with the smaller of the square unit/length but from Figure 5.6 it is observed that the more smaller is the square unit, the more accurate our proposed algorithms would be. For example, if we reduce the length of the square cell 2 times, IO access and CPU time increases on average 4 times in the hierarchical algorithm whereas in the grid algorithm IO access almost remains constant but CPU time rises sharply. However, in both algorithms, average approximation error decreases with the smaller of the square unit. For grid algorithm, it is 5% and for hierarchical algorithm it is more than 1%. The hierarchical algorithm gives on average 5 times more accurate meeting point compared to the grid algorithm.

## 5.2 Comparative Analysis

We show that the problem of finding the optimal meeting point in the obstructed space is NP-hard. Since processing an OOMP query in the obstructed space is an exhaustive search, as the search space is infinite and filled with obstacles. Thus the naive exact solution is not possible due to approximation error. The major challenges for an OOMP query is to refine the search space and compute the total obstructed distance with reduced processing overhead. Hierarchical algorithm is most appropriate in order to find out the result which is very much close to the optimal solution. Otherwise, grid algorithm is a good choice for finding the meeting point in real time with less accuracy level compared to hierarchical algorithm.

In all experiments, the CPU time and IO access of Grid algorithm is always lower than Hierarchical algorithm. The more smaller is the square unit threshold the more accurate meeting point is found in both algorithms. But the Hierarchical algorithm gives more accurate meeting point compared to Grid algorithm.

The reason behind lower cost of Grid algorithm is that, obstructed distance between the locations is computed heuristically. Thus accuracy is sacrificed but query processing overhead reduced. On the other hand, in the Hierarchical algorithm, obstructed distance between the locations is computed efficiently by exploiting geometric properties and hierarchical structure. As a result, query processing overhead increases with the increase of the number of the group members, obstacles and the search space but accuracy increases.

## Chapter 6

# Conclusion

Researchers have developed a number of algorithms for finding optimal meeting points in the Euclidean space and road networks that ignores the presence of obstacles. However, some research work has been done considering obstacles for other types of queries i.e. obstructed nearest neighbor queries, obstructed reverse nearest neighbor queries, continuous obstructed nearest neighbor queries, moving nearest neighbor queries and obstructed group nearest neighbor queries (OGNN). An OGNN query, returns the location of a (point of interest) POI from the given set of POIs, that minimizes the total obstructed travel distance with respect to the locations of the group members, whereas in case of an OOMP query a meeting point does not need to be at the location of a POI, it can be anywhere in the obstructed space except the areas of obstacles. Therefore, the OOMP query is much more difficult than the OGNN query due to its infinite search space.

### 6.1 Contribution

We summarize our key contributions in this thesis are as follows:

- We developed the first comprehensive solution to find the optimal meeting point in presence of obstacles which we call obstructed optimal meeting point queries (OOMP). Our proposed two algorithms find optimal meeting locations which minimizes the total obstructed distance of all the group members.
- To identify the optimal meeting point, computing the total obstructed distance for every point in the search space would incur extremely high processing overhead as finding the obstructed distance between two locations is an expensive computation. Thus, the major challenges for

an OOMP query is to refine the search space and compute the total obstructed distance with reduced processing overhead. We exploit geometric properties and hierarchical structure to develop techniques to refine the search space. In addition, we developed efficient technique to compute the total obstructed distance. Our technique reduces the number of obstacles retrieved from the database, and does not retrieve the same obstacle multiple times from the database to compute multiple individual obstructed distances required for computing a total obstructed distance.

- The obstacles are stored on the database using an indexing method. Processing an OOMP query in the obstructed space is an exhaustive search, as the search space is infinite and filled with obstacles. We develop an efficient hierarchical algorithm for processing OOMP queries, which recursively refines the search space in order to minimize the number of retrieved obstacles from the database and reduce the number of total obstructed distance computation using geometric properties. The query processing overhead increases with the increase of the number of group members, obstacles and the search space. Thus we also develop a grid algorithm for processing OOMP queries in real time. The grid algorithm finds the meeting location which incurs reduced processing overhead but accuracy is sacrificed.
- Our experimental results using real datasets are used in the performance analysis of our algorithms for different parameters. We compare our proposed two OOMP algorithms. We find that, the hierarchical algorithm incurs higher CPU time and IO cost than the grid algorithm, but in the grid algorithm we need to sacrifice accuracy. The reason behind lower cost of grid algorithm is that, here expensive obstructed distance computation techniques are avoided and the search space is turned into a number of binary square grids, which are candidates of optimal meeting locations. For this reason, grid algorithm compared to hierarchical algorithm performs better than in terms of CPU time and IO cost but the hierarchical algorithm finds more accurate meeting location than grid algorithm.

## 6.2 Future Work

This research work motivates several directions for future research. In this thesis, we have only considered static obstacles and query points. We have a plan to study OOMP queries considering



---

moving obstacles (i.e, vehicles in pedestrians walking path). As a future work, we intend to investigate OOMP queries where the query points i.e, the group of users can move. Preserving the privacy of the group members while answering OOMP queries can be another field of study for our future works.

# References

- [1] Reuven Chen. Location problems with costs being sums of powers of euclidean distances. *Computers & Operations Research*, 11(3):285–294, 1984.
- [2] Reuven Chen. Solution of location problems with radial cost functions. *Computers & Mathematics with Applications*, 10(1):87–94, 1984.
- [3] Leon Cooper. An extension of the generalized weber problem. *Journal of Regional Science*, 8(2):181–197, 1968.
- [4] Lawrence M. Ostresh. The multifacility location problem: Applications and descent theorems. *Journal of Regional Science*, 17(3):409–419, 1977.
- [5] Da Yan, Zhou Zhao, and Wilfred Ng. Efficient processing of optimal meeting point queries in euclidean space and road networks. *Knowledge and Information Systems*, 42(2):319–351, 2015.
- [6] Da Yan, Zhou Zhao, and Wilfred Ng. Efficient algorithms for finding optimal meeting point on road networks. *PVLDB*, 4(11):968–979, 2011.
- [7] Shivendra Tiwari and Saroj Kaushik. Scalable method for k optimal meeting points (k-omp) computation in the road network databases. In *Databases in Networked Information Systems - 8th International Workshop, DNIS. Proceedings*, pages 277–292, 2013.
- [8] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numer. Math.*, 1(1):269–271, December 1959. ISSN 0029-599X. doi: 10.1007/BF01386390. URL <http://dx.doi.org/10.1007/BF01386390>.
- [9] Jun Zhang, Dimitris Papadias, Kyriakos Mouratidis, and Manli Zhu. Query processing in spatial databases containing obstacles. *International Journal of Geographical Information Science*, 19(10):1091–1111, 2005.

- 
- [10] Yunjun Gao, Jiacheng Yang, Gang Chen, Baihua Zheng, and Chun Chen. On efficient obstructed reverse nearest neighbor query processing. In *ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 191–200, 2011.
- [11] Yunjun Gao, Baihua Zheng, Gang Chen, Chun Chen, and Qing Li. Continuous nearest-neighbor search in the presence of obstacles. *ACM Transactions on Database Systems (TODS)*, 36(2):9, 2011.
- [12] Chuanwen Li, Yu Gu, Fangfang Li, and Mo Chen. Moving k-nearest neighbor query over obstructed regions. In *APWEB*, pages 29–35. IEEE, 2010.
- [13] Hu Xu, Zhicheng Li, Yansheng Lu, Ke Deng, and Xiaofang Zhou. Group visible nearest neighbor queries in spatial databases. In *WAIM*, pages 333–344, 2010.
- [14] Yunjun Gao, Baihua Zheng, Gencai Chen, Wang-Chien Lee, Ken C. K. Lee, and Qing Li. Visible reverse k-nearest neighbor query processing in spatial databases. *IEEE Trans. Knowl. Data Eng.*, 21(9):1314–1327, 2009.
- [15] Yunjun Gao, Baihua Zheng, Gencai Chen, Qing Li, and Xiaofa Guo. Continuous visible nearest neighbor query processing in spatial databases. *VLDB J.*, 20(3):371–396, 2011.
- [16] Nusrat Sultana, Tanzima Hashem, and Lars Kulik. Group nearest neighbor queries in the presence of obstacles. In *ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 481–484, 2014.
- [17] Nimrod Megiddo and Kenneth J Supowit. On the complexity of some common geometric location problems. *SIAM journal on computing*, 13(1):182–196, 1984.
- [18] Nusrat Sultana, Tanzima Hashem, and Lars Kulik. Group nearest neighbor queries in the presence of obstacles. In *ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 481–484. ACM, 2014.
- [19] Amir Beck and Marc Teboulle. 1 gradient-based algorithms with applications to signal recovery problems, 2009.
- [20] Jack Brimberg and Robert F. Love. Global convergence of a generalized iterative procedure for the minisum location problem with lp distances. *Operations Research*, 41(6):1153–1163, 1993.

- 
- [21] G.O. Wesolowsky. Location problems on a sphere. *Regional Science and Urban Economics*, 12(4):495 – 508, 1982.
- [22] Stephen Boyd and Lieven Vandenbergh. *Convex Optimization*. Cambridge University Press, New York, NY, USA, 2004. ISBN 0521833787.
- [23] Zhengdao Xu and Hans-Arno Jacobsen. Processing proximity relations in road networks. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, pages 243–254, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0032-2.
- [24] Franco P. Preparata and Michael Ian Shamos. *Computational Geometry - An Introduction*. Springer, 1985. ISBN 3-540-96131-3.
- [25] Ali Reza and Zarei Mohammad Ghodsi. Efficient computation of query point visibility in polygons with holes. In *In Proceedings 21st Annual Symposium on Computational Geometry*, pages 6–8, 2005.
- [26] Takao Asano, Tetsuo Asano, Leonidas J. Guibas, John Hershberger, and Hiroshi Imai. Visibility of disjoint polygons. *Algorithmica*, 1(1):49–63, 1986.
- [27] S Suri and J O’Rourke. Worst-case optimal algorithms for constructing visibility polygons with holes. In *Proceedings of the Second Annual Symposium on Computational Geometry, SCG ’86*, pages 14–23, New York, NY, USA, 1986. ACM. ISBN 0-89791-194-6.
- [28] 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.
- [29] Boaz Ben-Moshe, Olaf Hall-Holt, Matthew J. Katz, and Joseph S. B. Mitchell. Computing the visibility graph of points within a polygon. In *Proceedings of the Twentieth Annual Symposium on Computational Geometry*, pages 27–35, New York, NY, USA, 2004. ACM. ISBN 1-58113-885-7.
- [30] Takao Asano, Tetsuo Asano, Leonidas J. Guibas, John Hershberger, and Hiroshi Imai. Visibility-polygon search and euclidean shortest paths. In *FOCS*, pages 155–164, 1985.
- [31] Tomás Lozano-Pérez and Michael A. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Commun. ACM*, 22(10):560–570, 1979.

- 
- [32] Ru-Mei Kung, Eric N. Hanson, Yannis E. Ioannidis, Timos K. Sellis, Leonard D. Shapiro, and Michael Stonebraker. Heuristic search in data base systems. In *Expert Database Workshop*, pages 537–548, 1984.
- [33] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numer. Math.*, 1(1):269–271, December 1959.
- [34] Jun Zhang, Dimitris Papadias, Kyriakos Mouratidis, and Zhu Manli. Query processing in spatial databases containing obstacles. *International Journal of Geographical Information Science*, 19(10):1091–1111, 2005.
- [35] Chenyi Xia, David Hsu, and Anthony K. H. Tung. A fast filter for obstructed nearest neighbor queries. In *BNCOD*, pages 203–215, 2004.
- [36] Yu Gu, Ge Yu, and Xiaonan Yu. An efficient method for k nearest neighbor searching in obstructed spatial databases. pages 1569–1583, 2013.
- [37] Yunjun Gao and Baihua Zheng. Continuous obstructed nearest neighbor queries in spatial databases. In *SIGMOD Conference*, pages 577–590, 2009.
- [38] Sarana Nutanong, Rui Zhang, Egemen Tanin, and Lars Kulik. V\*-knn: An efficient algorithm for moving k nearest neighbor queries. In *ICDE*, pages 1519–1522, 2009.
- [39] Yunjun Gao, Baihua Zheng, Wang-Chien Lee, and Gencai Chen. Continuous visible nearest neighbor queries. In *EDBT*, pages 144–155, 2009.
- [40] Yunjun Gao, Baihua Zheng, Gencai Chen, Wang-Chien Lee, Ken C. K. Lee, and Qing Li. Visible reverse k-nearest neighbor queries. In *ICDE*, pages 1203–1206, 2009.
- [41] Yafei Wang, Yunjun Gao, Lu Chen, Gang Chen, and Qing Li. All-visible-k-nearest-neighbor queries. In *DEXA (2)*, pages 392–407, 2012.
- [42] Dimitris K. Agrafiotis and Victor S. Lobanov. An efficient implementation of distance-based diversity measures based on k-d trees. *Journal of Chemical Information and Computer Sciences*, 39(1):51–58, 1999.
- [43] Jon Louis Bentley. Multidimensional binary search trees in database applications. *IEEE Trans. Software Eng.*, 5(4):333–340, 1979.

- 
- [44] Mark H. Overmars and Jan van Leeuwen. Dynamic multi-dimensional data structures based on quad- and  $k$ - $d$  trees. *Acta Inf.*, 17:267–285, 1982.
- [45] Murat Demirbas and Xuming Lu. Distributed quad-tree for spatial querying in wireless sensor networks. In *ICC*, pages 3325–3332, 2007.
- [46] K. Yamaguchi, T. Kunii, K. Fujimura, and H. Toriya. Octree-related data structures and algorithms. *IEEE Computer Graphics and Applications*, 4(1):53–59, 1984.
- [47] Timos K. Sellis, Nick Roussopoulos, and Christos Faloutsos. The  $r+$ -tree: A dynamic index for multi-dimensional objects. In *VLDB*, pages 507–518, 1987.
- [48] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD Conference*, pages 47–57, 1984.
- [49] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The  $r^*$ -tree: An efficient and robust access method for points and rectangles. In *SIGMOD Conference*, pages 322–331, 1990.
- [50] Visibility Graph. [http://en.wikipedia.org/wiki/Visibility\\_graph](http://en.wikipedia.org/wiki/Visibility_graph).
- [51] Tomás Lozano-Pérez and Michael A. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Commun. ACM*, 22(10):560–570, 1979.
- [52] Der-Tsai Lee. *Proximity and Reachability in the Plane*. PhD thesis, Champaign, IL, USA, 1978. AAI7913526.
- [53] Emo Welzl. Constructing the visibility graph for  $n$ -line segments in  $o(n^2)$  time. *Inf. Process. Lett.*, 20(4):167–171, 1985.
- [54] Subir Kumar Ghosh and David M. Mount. An output-sensitive algorithm for computing visibility graphs. *SIAM J. Comput.*, 20(5):888–910, 1991.
- [55] John Kitzinger. The visibility graph among polygonal obstacles: a comparison of algorithms.
- [56] Yufei Tao, Dimitris Papadias, and Qiongmao Shen. Continuous nearest neighbor search. In *Proceedings of the 28th International Conference on Very Large Data Bases, VLDB '02*, pages 287–298. VLDB Endowment, 2002.

- 
- [57] germanyDataset. Spatial (Geographical) Datasets In 2D Space Germany. <http://chorochronos.datastories.org/>.