Abstract

Spatial Alarms are an important class of personalized location based services (LBSs) that are triggered by a specific location of a moving user, instead of time. A pedestrian may plan to buy a medicine from a pharmacy while walking through a city or a tourist may want to have the dinner at a restaurant while taking a scenic walk at a unfamiliar place. Spatial alarms enable users to know if a point of interest (POI) such as a pharmacy or a restaurant comes within a specific range of the pedestrian. In this thesis, we introduce an efficient approach to evaluate spatial alarm queries in the obstructed space. Existing work in this area has focused mainly on the Euclidean space and road network models. However, the movement of a pedestrian is restricted with many obstacles like buildings, fences or vehicles. To trigger the spatial alarm for a pedestrian, an LBS needs to continuously check whether the specific type of POI is within the specific range of the current location of the pedestrian, which incurs high query processing overhead. We exploit geometric properties to refine the POI and obstacle search region in the total space and develop a technique to identify the area, where a pedestrians movement does not need to retrieve any new POI or obstacle from the LSP. Our approach avoids the retrieval of same POIs and obstacles from the LSPs databases. We perform extensive experiments using real data sets and show that our approach is significantly faster and requires less IO than a nave approach.

Chapter 1

Introduction

The high availability of smart phones has led to the proliferation of location based services. Starting from static LBS (Location Based Service) such as finding the nearest pharmacy for a user's location, now-a-days LBSs are tailored for moving users. According to many, the next step in location based services is Spatial Alarms. Many believe this particular feature is going to dominate the future mobile-phone computing systems. Spatial alarms are an extension of time-based alarms. It is, however, triggered by a specific location irrespective of time."Remind me if I'm within 100 meters of a pharmacy" is a possible example of a spatial alarm. It is a personalized location based service which can vary from user to user.

It is noteworthy that spatial alarms are quite dissimilar to spatial range query. Spatial alarms are based on a fixed location thus applying the techniques that are used in answering spatial range queries is both inefficient and wasteful for the two dominating reasons, Firstly, in spatial range query as the user is the main point of interest, continuous re-evaluation of her location is needed in case of a mobile user. In contrast, spatial alarms need only be re-evaluated when the user is approaching a specific location. Secondly, in spatial alarm, the main point of interest is a static location. So the user's location is not relevant at all times. It is quite clear that applying spatial range query techniques in evaluating spatial alarms is going to result in wastage of resources. If we start to evaluate spatial alarms as soon as the user is on the move even if the user is far away from her desired location our efforts will be futile.

1.1 Problem Setup

Existing research has categorized spatial alarms into three types: Public, Shared and Private. Public alarms are alarms which are active for every user within the system, such as an alarm must be sent to everyone within 100 meters of a building on fire. Private alarms are user defined alarms which can be viewed by the user, such as a user might set an alarm to alert her if she is within 100 meters of her favorite coffee shop. Shared alarms are shared between specific groups of people. In the previous example if a user chooses to share the alarm for the coffee shop with some of her friends it becomes a shared spatial alarm. In [**?**] spatial alarms has been categorized into three different types: 1) Moving subscriber with Static target, 2) Static subscriber with Moving target 3) Moving subscriber with Moving target. In this paper we are only considering the first type, that is Moving subscriber with Static target. In [**?**] spatial alarm region have been approximated by rectangular bounding box, in our approach we are considering the spatial alarm region as a circle with radius r.

Obstructed Space Path Problem [**?**] denotes the problem of finding the shortest route between two query-points in Obstructed Space where non-intersecting 2D polygons represent *obstacles* and where the route does not traverse through any obstacles. The length of the Obstructed route between points a and b is called the *Obstructed Distance* between a and b, denoted by $dist_o(p_i, q)$.

A **Spatial Alarm Query in Obstructed Space** is formally defined as follows: Given a moving query point q and a range r for an alarm, a Spatial Alarm Query returns $\forall p_i \in P = \{p_1, p_2, p_3 ... p_n\}$ which have $dist_o(p_i, q) < r$

1.2 Preliminaries

Spatial alarms are location-based, user-defined triggers which will possibly shape the future mobile application computations. They are distinct from spatial range query and do not need immediate evaluation after the user has activated them. The spatial alarm evaluation strategies are judged based on two features, High accuracy and High system scalability. High accuracy refers to the quality

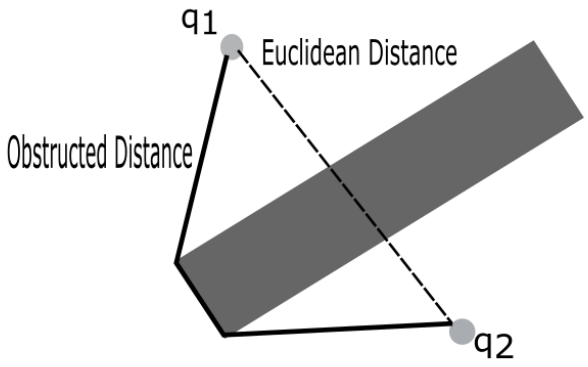

Figure 1.1: Obstructed Distance

that guaranties no alarms are missed. And High scalability is the feature that ensures that the system can adapt to a large number of spatial alarms.
In this paper, We propose a novel approach to evaluate spatial alarms in obstructed space which ensures both High accuracy and High scalability.

We define three different type of regions: *Known Region*,*Reliable Region* and *Safe Region*

Known Region:
The region within which all POIs (alarms) and obstacles are known to the client We define two different known regions for the POIs and the obstacles. Both of the known regions are represented by a parabola whose focus point is the users location q,with the equation $y^2 = 4ax$ where $a = mr$ which is bounded by a straight line.

Reliable Region:

Within which region, no further query to the server has to be done to compute a consistent set of answers, that is termed as a reliable region. The reliable region is also a parabolic region bounded by a straight line,where each bounding point of the parabolic curve is at a distance r from the known regions parabolic curve. $\forall p_i = (x, y)$ in the known region, $\forall p_j = (x_r, y_r)$ in the reliable region, $dist_E(p_i, p_j) \geq r$ By this definition no further queries to the server has to be done to compute a consistent set of answers. Because, for any $q = p_j$ $dist_E(q, p_j) \geq r$ where p_j is a point on the boundary of the reliable region. And for all other points p_i inside the reliable region $dist_E(q, p_i) > r$

Safe Region:

A safe region is the region located inside reliable region within which the answer set of POIs remains unchanged for a moving client. We will denote the radius of the safe region as r_{safe}. Given the user's previous location P_1 and the current location P_2, if $(P_1 - P_2) < r_{safe}$, then no recalculation is needed to compute the answer. If the safe region surpasses the reliable region at some points

The key idea of our approach is to calculate a dynamic *safe region*, within which no computation has to be done to provide an accurate alarm trigger. We will use an R-tree structure to index both obstacles and POI's in our approach. Our spatial alarm processing system has two different modes for efficient and effective processing of spatial alarms,namely, Bandwidth saving mode and Computational Cost Saving mode.

We assume that the system is based on a client-server architecture and the POIs as well as the obstacles are stored using independent R-trees at the server. We also assume that all users have access to some sort of localization service such as GPS or Wi-Fi that queries the server with the client's pinpoint location. Here, the client application is a thin-weight application that communicates with the server on any specified event to retrieve necessary information about POIs and the obstacles. We assume that the user can use any device such as smart-

Table 1.1: Symbol Table

Symbols	Meaning
P	Set of POIs
O	Set of Obstacles
q	Location of user
r	Alarming range
V_G	Visibility Graph
$dist_E$(p,q)	Euclidean distance between points p and q
$dist_O$(p,q)	Obstructed distance between points p and q
r_{safe}	Radius of safe region
m	Real number in the range $[2, \infty)$
n	Real number in the range $[1, \infty)$
θ_i	Angle between consecutive path segments
S	Users path history as a set of straight lines
(m_i, c_i, l_i)	A line with slope m_i, intercept c_i and length l_i

phones or PDA.

1.3 Contributions

Our approach accounts for both accuracy and efficiency by focusing on (1) No alarms being missed in user's proximity (2) Avoiding wasteful computation in client side (3) Minimizing data transfer between server and client. In summary the our main contributions are:

- We introduce an efficient algorithm for evaluation of spatial alarm queries in obstructed space.
- We provide a spatial alarm evaluation system for mobile user.
- We provide an algorithm to calculate a dynamically changing region to accurately evaluate spatial alarm queries without any computation.
- We provide an extensive experimental analysis to compare the accuracy of our approach with other naive approaches based on different parameters.

1.4 Thesis Organization

The next chapters are organized as follows. In Chapter 2, related works are discussed. In Chapter 4 and 3, we present a naive approach and our approach, respectively, to evaluate Spatial Alarm in Obstructed Space queries. Chapter 5 shows the experimental results for our proposed algorithm. Finally, in Chapter 6, we conclude with future research directions.

Chapter 2

Related Works

In this chapter, we give a description of previous works related to this thesis. Spatial alarm processing technique in Road Network and Euclidean Space has been extensively studied in recent years.In this chapter, we first show the related works on Spatial Alarm queries in Road Networks in 2.1,then we show the related works on different queries in the Obstructed Space in the section 2.2 finally in **??** we show the effectiveness of R-tree as an data structure.

2.1 Spatial Alarm Queries

Extensive research has been performed and various effective algorithms have been proposed [**?**],[**?**],[**?**] to process spatial alarms in Euclidean space and road network in recent years. Euclidean space considers the straight line distance between two points irrespective of obstacles between them on the other hand in road networks navigation is limited along predefined roads.
Again, comprehensive research [**?**] has been conducted to make spatial alarm evaluation energy-efficient and effective in road networks.
However, to the best of our knowledge no research work has yet been published on the topic of spatial alarms in obstructed space. In this section we present the related research work in the computation of spatial alarms in the following subsections.

In [?] the authors argue that spatial alarm queries are best approximated by road network scenarios as the clients movement is more likely to follow some specific road network. They show that the road network distance-based spatial alarm is best modeled using road network distance such as segment length-based an travel time-based distance. in their opinion, a road network spatial alarm is a star-like subgraph centered at the alarm target. [?] introduced road network based spatial alarm and provided several optimization techniques to make spatial alarm computation in road network efficient.

Murugappan et al. in [?] have studied a middleware architecture for energy efficient processing of spatial alarms on mobile clients, while maintaining low computation and storage costs. Their research on spatial alarms provides two systematic methods for minimizing energy consumption on mobile clients. They introduce the concept of safe distance the number of unnecessary mobile client wakeups for spatial alarm evaluation.their approach enables mobile clients to sleep for longer intervals of time in the presence of active spatial alarms. Murugappan et.al have studied the computation of spatial alarm using a middle ware architecture in the road networks. In [?] a safe region based approach has been introduced to effectively compute spatial alarms. The authors argue that, the conventional computation of spatial alarm queries using server centric architecture can be improved upon by introducing a distributed safe region based approach. The safe region based approach delegates some of the computational overhead to the client and results in optimal usage of resources. In this thesis , we adapt the concept of safe region and the distributed architecture in processing spatial alarms. Bamba et al. in [?] have given three different approaches to compute the safe region. However, none of them are tailored to be used in obstructed space. In each of the above mentioned research works efficient and effective techniques have been studied to process spatial alarm queries in road networks and euclidean space. However, to the best of our knowledge, no research work has yet been published for the computation of spatial alarms in the obstructed space.

2.2 Different Queries in Obstucted Space

In this section we study various spatial queries that have been extensively researched in the obstructed space. Some of these queries have very little in common with spatial alarm queries, nevertheless we study the obstructed distance computation techniques used in them. Papadias et al. in [?] have provided approaches to compute several important spatial alarm queries in the obstructed space, which include range search, nearest neighbors, e-distance joins and closest pairs. They use R-tree for indexing both data points and obstacles. Zhange et al in [?] have introduced reverse nearest neighbour queries in obstructed space.Given a datapoint q, a reverse nearest neighbor finds all the points/objects that have q as their nearest neighbor. In [?] the authors have introduced obstructed reverse nearest neighbour. They use effective pruning heuristics (via introducing a novel boundary region concept).However, spatial alarm queries and RNN queries are distinct from each other. A spatial alarm query returns all the points p that are within an alarm radius r of q.It can be argued that spatial alarm queries can be processed using RNN query solving algorithms, but it will not be as effective and efficient as a query that is specifically tailored to handle spatial alarms. Nutanong et al. in [?] have studied moving knn queries in detail.Mknn queries can be defined as follows, given a set of datapoints P and a moving query point q,an MkNN query retrieves the k nearest data points of q from P. As the client is moving the answer set may need to be recomputed as the client moves. They have used the concept of safe region,reliable region and known region in their paper. They use the safe region to reduce unnecessary checks for knns near the user. They have also introduced known region and reliable region to find the safe region for a specific location of a client. They have used a unique V* diagram approach to compute knns in their paper.The V* diagram is constructed using $(k + x)^{th}$ nns' of a user.The V* diagram computes two types of safe regions :Safe region with respect to a data point and fixed rank region. Then both of them are intrigated to form a intrigated safe region. Many qualities of[?] have been adapted in [?] which is discussed next. In this thesis we also use the terms, safe region,reliable region and known region. But our thesis is distinct from [?] on these points

- We use the known region and reliable region not only to help compute

the safe region, but also to reduce duplicate data transfer and number of server queries made.

- We use the concepts of these regions to efficiently compute spatial alarms. Here we would like to point out, that although spatial alarm queries can be solved using an extension of moving mknn queries , those extensions will result in wastage of resources and efficiency , as they are not specifically tailored for spatial alarms.

- We have used the above mentioned regions with respect to obstacles which is not considered in [**?**]

Li et al. in [**?**] have proposed an algorithm to efficiently compute moving Knn queries with respect to obstacles. However in their paper, Li et al. have proposed a safe region based approach, so that no recomputation is needed as long as the user is within the safe region. In [**?**] no predefined trajectory of client is assumed. The algorithm proposed in [**?**] can be summarized as follows:

- At the start of the query the algorithm first computes a list of k+x nearest data points of q, where the x extra data point serve as a cache to reduce the number of safe region recomputation.

- After every movement of client the algorithm uses the *obstructed safe region* and *the obstructed fixed-rank region* to determine whether a safe region recomputation is required and which data points need to be accessed for the recomputation.

Obstructed safe region w.r.t. datapoint is a region where the movement of q will not cause p to be removed from the k + x nearest neighbors of q.
Obstructed fixed rank region The obstructed fixed-rank region of a list L_{k+x} of k + x data points ordered ascendingly by their distances to the query point at q_b,is a region in an obstructed space that, when q moves inside the region, the distance rank of the data points in L_{k+x} to q is fixed.

In this paper they have also formulated an *obstructed known region* with the help of an *obstructed disk*. When the query point q is at position q_b and its $(k + x)^{th}$ nearest data point is z, the obstructed known region of q, is defined

as a region where the obstructed distance from any point t to q_b is less than or equal to the obstructed distance of z to q_b. An obstructed disk in obstructed space is a disc where the points distances to q_b are less than or equal to r.

In this paper the concept of safe region and fixed rank region have been adapted from [**?**]. In this thesis we have also adapted some of our key concepts from the same and manipulated them for using in the obstructed space. However, it is noteworthy that our approach and [**?**] are distinct from each other. Apart from the obvious fact that our thesis and [**?**] deal with two different type of spatial queries all together we would also like to point out two principle differences between them:

- Firstly, we adapt the concept of known region and reliable region from [**?**] in obstructed space. In this thesis, two different known region for POIs and Obstacles have been introduced to reduce duplicate and frequent data retrieval from the server. Our definition of known region is distinct from [**?**].
- Secondly, We have developed an efficient approach to accurately answer the spatial alarm queries in obstructed space by predicting the user's next direction of movement, a prospect that has not been considered in [**?**].

In [**?**] an efficient approach to solve the obstructed group nearest neighbour query is introduced.A GNN(Group Nearest Neighbor) query is a query that enables a group of users to meet at a point with minimum aggregate travel distance. Obstructed distance computation is a inexplicable part of any query in obstructed space. To find the obstructed distance between two points we need the visibility graph [**?**]. Usually, visibility graph construction is an $O(n^2)$ time consuming algorithm. To overcome this particular hurdle, we have adapted the approach of incrementally constructing the visibility graph used in both [**?**] and [**?**]. In this approach only the datapoints and obstacles that are absolutely necessary to our query is kept in the visibility graph. At first an initial visibility graph is constructed and then only the obstacles and that are relevant in the obstructed distance computation of the active datapoint set are kept in the visibility graph, all other obstacles are removed. This initial visbility graph is

incremented or decremented with necessary datapoints and obstacles as per our query.

B-tree [**?**] is a popular data structure that is often used in file systems and database systems because of it's logarithmic insertion and deletion running times and as it's child nodes can be accessed at the same time. However, B-tree can not store new type of data (i.e. geometrical data, multi-dimensional data). So Guttman [**?**] provided R-Tree for this task. R-tree has the following main properties

- R-Trees represent data as a minimum bounding rectangle MBR which enables it to store any dimensional data
- It is height balanced.
- Each node bounds it's children. A node can have many objects in it.
- The leaves point to the actual objects which are stored on the disk.

Chapter 3

Naive Solution

To compare the efficiency of our approach, we present a straightforward solution for processing Spatial Alarms in Obstucted Space. The naive approach is a straight forward approach with region computation. This approach searches for a new alarm in the known region as soon as the client changes it's position.

In Section 3.1, we give an overview of the naive solution. Section 3.2 presents the algorithm of the solution.

3.1 Overview

In this naive approach, the server is frequently queried to get the POIs and obstacles within the Euclidean distance r. Then the client checks the obstructed distance of each POI to be within r obstructed distance and fire alarm if that is reached.

3.2 Algorithm

In the Algorithm 4, the GetAllPOI(q, r) function populates the set P of POIs within the radius r centring the query point (the client's current location) q. Similarly, GetObstacleSet(q, r) function returns the set of all obstacles within the radius r centring q.

MakeVisGraph(P, O) function returns the *visibility graph* V_G with the set of POIs P and the set of obstacles O.

Here, a **Visibility Graph** is a graph $V_G(V, E)$ where each $v \in V$ is either a POI or a data-point and for each $(u, v) \in E$, there is an edge e between u and v if and only if it does not intersect with any obstacles *i.e.* u and v are *visible* to each other along the edge e.

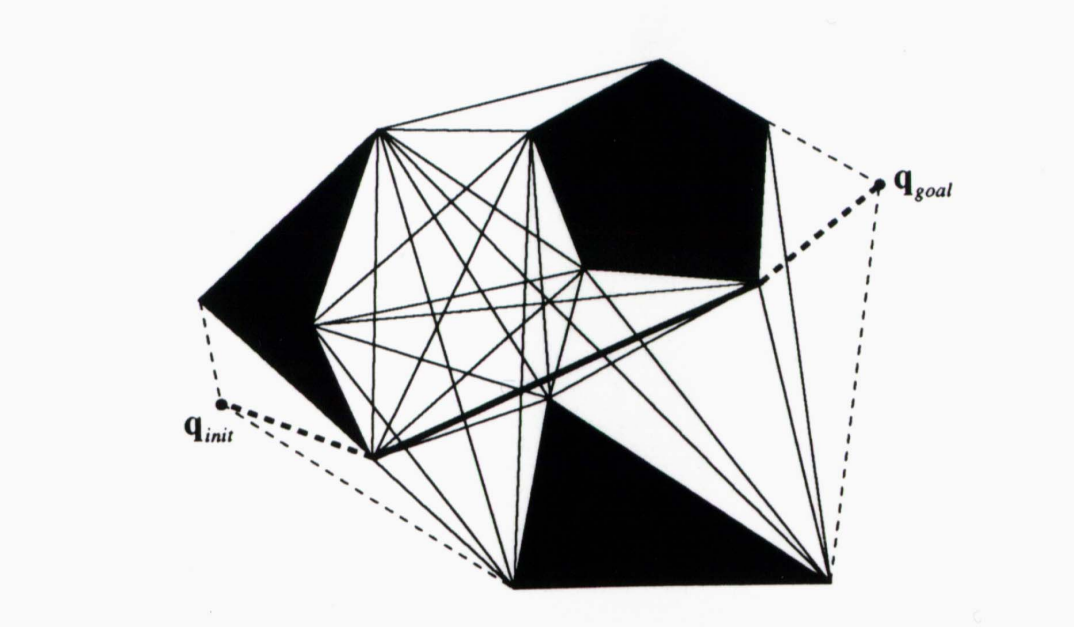

Figure 3.1: Visibility Graph

MakeAnswerSet(q, V_G) function returns a heterogeneous data-model consisting of its all parameters to be used by the caller client-side program.

Algorithm 1: GetAlarmables$(q,\ r\ ,\ A_{prev})$

Input : Query point q and the search radius r

Output: The visibility graph V_G

1 $P \leftarrow$ GetAllPOI(q, r)

2 $O \leftarrow$ GetObstacleSet(q, r)

3 $V_G \leftarrow$ MakeIncrementalVisGraph(P, O, A_{prev}, V_g)

4 $D_O \leftarrow$ getAllObstructedDist(q, V_g) **return** MakeAnswerSet(q, V_G, D_O)

The following method is triggered on any change of the user's location by the system checking whether to give any alarm to the client or not along with the check of necessity to fetch more POI and obstacle when the client goes outside of the farthest POI's alarming zone. Here, the function AlarmUser(p_i) triggers an alarm to the user since the respective POI p_i is reached and also marks p_i to be reached in the set of POIs P.
The inputs to this algorithm are the current client-location q and the answer set A consisting of the region's center q,POI set P, obstacle set O, and the visibility graph V_G.

Algorithm 2: UpdateClient(q, A)

Input : Client's current location q, latest answer set A

```
1 foreach p_i ∈ P do
2     if dist_O(q, p_i, V_G) > p_i.u then
3         AlarmUser(p_i)
4 if dist_E(A.q, q) > A.d_u then
5     GetAlarmables(q, 100)
```

In this naive-approach, the visibility-graph is constructed more times than necessary to hold accuracy, each of which constructions requires $O(n^2)$ [**?**], where n = the number of edges of the obstacles. A huge overhead is also sufficed to make P and O sets using such procedure. Again, the Algorithm 4 can also be run much less time than in this approach, which is improvised in the later approach.

Chapter 4

Our Approach

During building up the main approach, it is observed that spatial alarm evaluation can be optimized using three key features:

- Firstly, reducing the number of device wake-ups;
- Secondly, reducing any re-computation;
- Thirdly, reducing the data communication overhead between the server and the client.

For the first strategy to be successful, we propose an algorithm in this section which will compute an optimal safe-region. The second optimization technique is realized by passing optimal parameters among different functions as well as between the client and the server, while the third one is achieved by passing minimal parameters between the client and the server and in some cases recomputing some values in each side.

However, the second and the third options have some conflicts in some cases and therefore cannot be achieved simultaneously. For this reason, we have separated some parts of our main approach within two different modes, namely - *Bandwidth Saving (BWS) Mode* and *Computational Cost Saving (CCS) Mode*. In Section 4.1, we give an overview of the computation of the regions neccessary for our approach. Section 4.2 presents the algorithm of the Bandwidth Saving Mode. Section 4.3 presents the algorithm of the Computational Cost Saving Mode.

4.1 Computation of the regions

In our approach, the known region is bounded by a parabola, whose focus is the user's location q, the standard equation of the parabola being $y^2 = 4ax$ or, $x^2 = 4ay$ according to the client movement history, where $a = mr$.

First, we estimate a direction vector for the user's next movement using previous path history of the user. We take this direction vector as the major axis of the parabola. The exposure of the parabola depends on m_p. If the user is likely to move along a straight line, the exposure of the parabola need not be very high and only a narrow width parabolic region's data is sufficient to compute the client's answer set efficiently. But if the user is likely to move away from the major axis of the parabola very much frequently, the exposure should be accordingly large enough to host more surrounding data as the client is observed to go towards any direction and have any surrounding data in the answer set. A function is defined which estimates the value of m_p from the changes in user's direction from his previous trajectory, which are assumed to be piecewise linear for any kind of movement and saved in our implementation as a finite set of vectors. However,the function will ensure that the minimum value of m_p will be 2, which in return defines a reasonable minimum bound on the width of the known region.

The parabola is bounded on the open side by a straight line parallel to the directrix of the parabola. The distance of this straight line from the user's current location as well as the focus of the parabola is dependent on the client's velocity towards the predicted direction or axis-vector of the parabola. Thus, we will position the bounding straight line of the known region at a distance nr from the focus, where the value of n_p is is a finite function of the client's velocity. Intuitively we can see that, if the user moves fast, this bound should be larger, as the user is likely to move through the region quickly. Thus n_p is proportional to the client's velocity v. We introduce an intuitive function in this thesis, which will compute the value of n_p, the minimum bound being 1.

There are two such parabolic known regions in our main approach - primary one for the POIs and an equal or bigger one for the obstacles. The known region

for the POIs is built as described above, whereas the region for the obstacles is intialized with the paraboallic known region of the POIs and expanded later for any collision of any obstacle. If the collision is with the perimeter of the parabola, then the value of m_o is increased, whereas the value of n_o is increased if the collision is with the bounding line of the region. This increment process goes on unless there is no collision with any retrieved obstacle and the known region boundary. If any POI is found within any newly expanded region of the parabola, then the known region of the POIs is again set to the current bounded prabollic region of the obstacles and the check continues further on with later increment of the known region of the obstacles for anymore collisions of any obstacle and the boundary of the bounded parabolic known region of the obstacles. So, in short, the known region of the POIs is a bounded parabolic subset of the bounded parabolic known region of the obstacles.

The computation of the reliable region is comparatively less complex once the known regions are finalzied. The reliable region is also a parabolic region inside the the known region of the POIs. The vertex of the parabola of the POIs' known region is moved r distance closer to the client's location, which is also the focus of the parabola of the POIs. According to the geometric property of parabola, other points on the perimeter of the reliable region parabola comes more than r distance closer to the major axis of the known region parabola, assuring that the client won't need to query the server for anymore POI or obstacle to compute the answer set accurately while within the reliable region.

Finally, the safe region is computed using the nearest unalarmed POI p, where maximum range of the safe region $= dist_E(p, q) - r$. The safe region is the intersection of the circle centered at q with radius $dist_E(p, q) - r$ and the bounded parabolic reliable region.

The above described regions are depicted in the Figure 4.1 with necessary labels.

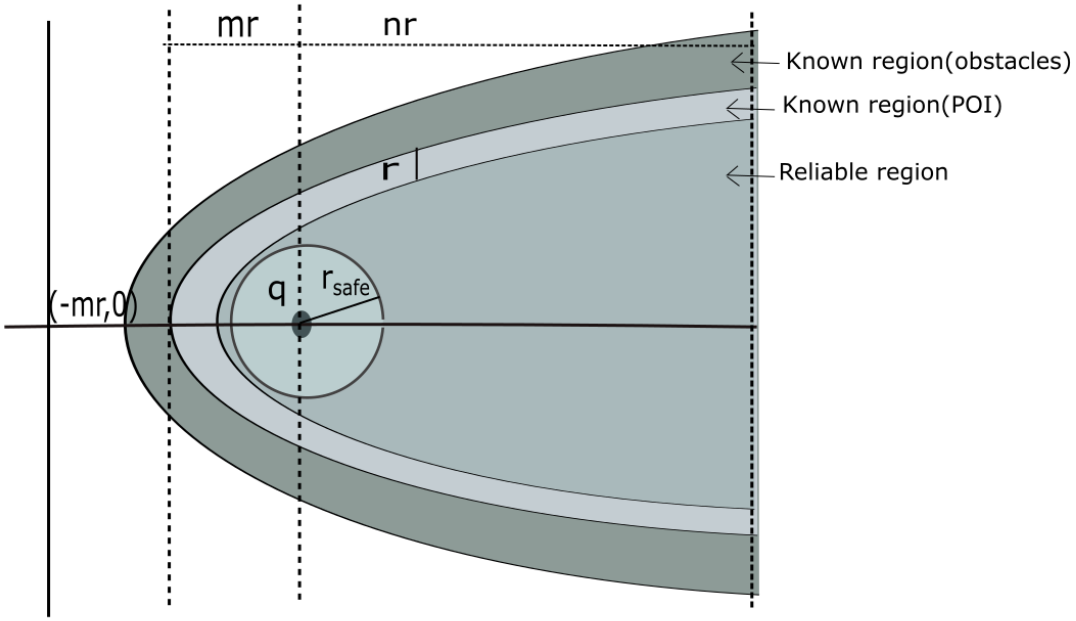

Figure 4.1: Different regions

4.2 Bandwidth Saving Mode

In this mode, the main focus is to reduce the bandwidth of communication between the server and the client. This mode is designed to operate in four parts - *client-initialization*, *the server query*, *alarm-configuration* and finally *update on any minimal amount of location change*. The Algorithms 3, 4, 5, 6 show the algorithmic-steps for these three parts respectively.

First ever entry point of our approach to the client is the Algorithm 3. The input to the algorithm is the client's current location q, alarm range r, current velocity v and movement history $S = \{(m_1, c_1, \vec{d_1}), (m_2, c_2, \vec{d_2}), (m_3, c_3, \vec{d_3}), ... (m_n, c_n, \vec{d_n})\}$ of that specific client. The next movement direction $\vec{s}_{Axis}$ is then predicted using S. The intuitive prediction used in our implementation is one kind of linear interpolation as the weighted average of the recent movement paths' slope (m) giving some value of the intercept (c) to predict the next direction towards the straight line, $y = mx + c$ for any specific (x, y) position of the client.

In the Algorithm 3, PREDICTDIRECTION(S) function returns a vector by inerpolating all the line segments given in a set S as the client's path history. During the linear interpolation, the oldest path is given the minimum weight, while the later ones get the incrementally higher weights iventually giving the latest path segment the maximum weight and predicting the client's current direction most likely to be towards the latest paths.
The function VARIANCEOFPATH($\vec{s}_{Axis}$, S) returns higher value for higher variation of the movement paths of the client from the currently predicted direction as from the movement history S. Inside this function, actually a biassed average of the distances of the client's point of change of path between two consecutive piecewise linear path segments is taken. The bias confirms that the returned value will meet the minimum bound of the value of m_p, which is 2 to ensure the accuracy and definition of the computed regions. In other words, this function returns a real number within the range [2, ∞) which is proportional to the variation of the client's path directions given the directed path segments in the set S. This value is assigned to the multiplier m_p to construct the parabola $y^2 = 4 * (m_p * r) * x$, since the more variation in the client's path requires more cross-section area of the known region parabola.
An intuitive logarithmic function is used to define the value of n_p =max($ln(e + v)$, $ln(e \times v)$).

After computing these values, a call to the Algorithm 4 contributes new POIs and obstacles to the current answer set. Finally, the reliable region π_r is computed as described in the sub-section 4.1 calling the function GETRELIABLEREGIONPARABOLA($A.$
where $A.\pi_p$ is the current bounded parabolic known region of the POIs.

The input to the Algorithm 4 is the current location of the client as well as the query point q, alarm range r, directed axis of the parabola $\vec{s}_{Axis}$, the last answer set A_{prev}, calculated variation of path m_p and the velocity dependent value n_p to bound the parabolic known region by a straight line. This algorithm makes the frequent queries more efficient, accurate and needs much less server communication.

The GETVERTEX(q, s_{dir}) returns the vertex point u of the parabola having

Algorithm 3: InitializeClient(q, r, v, S)

Input: Client's current location q, alarm range r, current velocity v and movement history S

1 $\vec{s}_{Axis} \leftarrow$ predictDirection(S)
2 $m_p \leftarrow$ varianceOfPath$(\vec{s}_{Axis}, S)$
3 $n_p \leftarrow \max(ln(e+v),\ ln(e \times v))$
4 $A \leftarrow A \cup$ getKnownRegionData$(q, r, \vec{s}_{Axis}, A, m_p, n_p)$
5 $\pi_r \leftarrow$ getReliableRegionParabola$(A.\pi_p, m_p, n_p, r)$
6 ConfigUpdate(q)

focus at q and the axis along the vector s_{dir}.

The GetObstacleSet$(q, m_o \times r, n_o)$ returns all obstacles within the parabola $y^2 = 4 \times (m_o \times r) \times x$ which is again bounded by a straight line at $(n_o \times r)$ distance from the focus q and perpendicular to the axis of the parabola.

In this algorithm, AttachToVisGraph(V_G, P, O) function adds the POIs and obstacles in the sets P and O respectively to the provided visibility graph V_G, so that minimal re-computation is needed.

The checkCollision(π_o, O) function returns 1 for collision of any obstacle with the parabolic permeter of the known region, 2 for collision of any obstacle with the bounding straight line and negative value for no collision of any obstacle in the queried set of the obstacles.

The output of the Algorithm 4 is an answer set A, which consists of the parabola of the known regions of POIs and obstacles (π_p and π_o respectively) and the set of all POIs within bounded parabola π_p. Since it is a bandwidth saving mode, the visibility graph is sent to the client as for incremental construction of he visibility graph in the next iteration.

The input to the Algorithm 5 is the current location of the client q and the answer set got from the Algorithm **??**. This algorithm is also responsible for triggering an alarm to the client if the client is within the alarming distance of any POI.

Algorithm 4: getKnownRegionData$(q, r, \vec{s}_{Axis}, A_{prev}, m_p, n_p)$

Input : Query point q, query radius r, axis of the parabola $\vec{s}_{Axis}$, the last answer set A_{prev}, m_p and n_p

Output: The answer set, $A \leftarrow \{q, \pi_p, \pi_o, P, V_G, D_O\}$

```
1  u ← getVertex(m_p × r, s_Axis)
2  s_dir ← getDirectrix(s⃗_Axis, u, q)
3  s_b ← getBoundingLine(s⃗_Axis, n_p × r)
4  π_p ← BoundedParabola(q, s_dir, s_b)
5  P ← getPoisWithinParabola(π_p)
6  π_o ← π_p
7  O ← getObstaclesWithinParabola(π_o)
8  f ← checkCollision(π_o, O)
9  while f > 0 do
10     if f == 2 then
11         n_o + +
12     else
13         m_o + +
14     P_p ← getPoisWithinParabola(π_p)
15     O_p ← getObstaclesWithinParabola(π_o)
16     if |P_p| > |P| then
17         P ← P_p
18         m_p ← m_o
19         n_p ← n_o
20         π_p ← π_o
21     O ← O_p
22     f ← checkCollision(π_o, O)
23 if A_prev.V_G == NULL then
24     A_prev.V_G ← constructInitVisGraph(P)
25 R_t ← getObstacleRtree()
26 D_O ← getObstructedDist(A_prev.visGraph, q, P, R_t)
27 return MakeAnswerSet(q, π_p, π_o, P, V_G, D_O)
```

Algorithm 5: ConfigUpdate(q, A)

Input : Query point q, answer set A

```
1 foreach p_i ∈ P do
2     p_i.d_O ← dist_O(q, p_i)
3     if p_i.d_O ≤ r then
4         AlarmUser(p_i)
5     else if p_i.d_E < r_safe then
6         r_safe ← p_i.d_E
7 return r_safe
```

In this algorithm, the visibility graph is computed using the answer sets P and O got as return of the **??** from the server side. The input to the algorithm 6 is the current location of the client q, the radius of the safe region(r_{safe}) and finally the already computed answer set A. The output of the algorithm is the minimum distance d_u to trigger this algorithm the next time.

Algorithm 6: UpdateOnLocChange(q, r_{safe}, π_r, A)

Input : q, r_{safe}, A

Output: d_u

```
1 if isOutsideReliableregion(q, A) then
2     A ← getKnownRegionData(q, d_q)
3     r_safe ← ConfigAlarm(q, A)
4 if d_q > r_safe then
5     r_safe ← ConfigAlarm(q, A)
6 return d_u ← r_safe
```

4.3 Computational Cost Saving Mode

The algorithms run in this mode almost similarly as in the "*Bandwidth Saving Mode*" with all the four parts - *client-initialization*, *the server query*, *alarm-configuration* and finally *update on any minimal amount of location change* as demonstrated in the Algorithms 3, 4, 5, 6.

However, the main difference with the previously descried mode from this mode is - the Algorithm 4 returning the computed V_G as another element of the answer set A from the server side and the algorithm 5 not reconstructing this V_G in the client side during running the Algorithm 5, whereas the Algorithm 6 remains all the same.

Therefore, the computation overhead for computing the V_G is saved in the client-side in cost of a one-time communication overhead in transferring the V_G in the Algorithm 4.

4.4 Proof of Correctness

No server query is needed while the user is inside the reliable region.

Proof. By the geometric property of parabola, it is known that every point on a parabola is equidistant from the focus and the directrix, that means, in the Figure 4.1, $SP = PM$ for the known region parabola and $SP' = P'M'$. Among all such points P and P', the vertex is the closest one to the focus.
Now, according to the computation of the reliable region, the vertex of the parabola is brought r distance closer to the client's location as well as the focus of the parabola of the known region. So, for any other point P on the perimeter of the known region and any other point P, on the perimeter of the reliable region, $dist_E(P, P') \geq r$. Again, by the *Euclidean Lower Bound (ELB)* property, $dist_O(P, P') \geq dist_E(P, P')$. Therefore, no POI outside the known region will be in the alarming range of the client and the client need not query the server for any more data point while inside the reliable region. $\mathcal{Q}.\mathcal{E}.\mathcal{D}.$

No computation is needed other than location calculation inside the safe region.

Proof. Since the safe region is a sub-set of the reliable region, so by the proof-1, no server query is needed to compute the answer set while the client is inside the safe region.

Now, by the construction of the safe region, the maximum Euclidean range of the safe region is, $r_{safe} = min_{\forall p \in P} dist_E(p, q) - r$. Therefore, even the next

closest to be alarmed POI will not be missed even if no calculation is done while the cleint is inside the safe region.

However, the client's location q must be regularly computed to check whether the safe region is crossed or not. Therefore, it is proved that no computation is needed other than location calculation inside the safe region. $\mathcal{Q}.\mathcal{E}.\mathcal{D}.$

Chapter 5

Experimental Study

5.1 Experiment Setup

To test and support our theoretical approach to compute spatial alarms in the obstructed space efficiently, we have done extensive experiments using both real and synthetic data-sets. In this chapter, we are going to present the validation and comparison of the proposed approach against the naive approach regarding various effective parameters

In this section, the detailed setup of the experiments is described as per the following sub-sections

5.1.1 Data Set Used

We have used both real and synthetic data-sets to evaluate our solution. In case of real data-sets, we have used obstacles and point of interests (POIs) of Germany. The obstacle set has 30674 minimum bounded rectangles (MBRs) of railway lines (rrlines). In our experiment, we assume obstacles to be presented by MBRs, but our algorithm can handle any type of obstacles. The POI set has 76999 MBRs of hypsography data (hypsogr). We assume data-points to be endpoints of the hypsography data. In this way, the POIs and obstacles are in the same plane which allows us to simulate a real-life scenario. We do not allow intersections between POIs and obstacles, neither do we allow duplicate POIs or obstacles. In case of synthetic data-sets, we have generated the obstacles and POIs from the real datasets following a uniform distribution. Before using in the real experiment, we have always normalized both real and synthetic dataset

in a $10,000 \times 10,000$ grid.
In our experiments, we have assumed only one type of POIs. However, our approach also can handle multiple types of POI.
Prior to running the main algorithms, two separate R-trees are built to store the POIs and the obstacles respectively from the used data-sets.

5.1.2 Sample Query Generation

We have simulated the movement of the client randomly in the runtime. During the experiments, we have assumed that any movement-path of the client can be synthesized as piecewise-linear, i.e., a set of directed straight lines. In the explicit form of any straight line, $y = mx + c$, for any client position (x, y), we have randomized the slope m giving some value of c each time. The direction of the client along this new straight line is also randomized to be either in forward or backward along the path, with a bias towards the forward direction, as the real world user-movement with a definite source and destination usually proposes.
After determining the clients new position along this path, in the naive approach, the algorithm 1 directly queries the server for POIs and obstacles within the clients alarming range. Alternately, in the main approach, the algorithm 6 checks the region-crosses and queries the server if necessary.
The clients velocity is also randomized within a certain range to give a new position of the client along the current direction in the next iteration, which again generates a new server-query accordingly.
Meanwhile, the direction of the client is predicted in the main approach from the latest set of piecewise linear paths in the clients movement history. This prediction procedure has already been described in the Algorithm section.

5.1.3 Measurement of the performance parameters

In our experiments, we vary the following query parameters:
(i) Clients Velocity Range
(ii) the Alarm Range r

(iii) the size of data sets (synthetic data set)
. Table 5.1 summarizes the parameter values used in our experiments. In all experiments, we estimate I/O accesses and the query processing time to measure the efficiency of our algorithms. In each set of experiments, we run the experiment for 100 queries and present the average result.

Parameter	Values	Default
Clients Movement lenght(v) Range (Unit)	500,1000,1500,2000	2000
Alarm Range r	60, 80, 100, 150	150
Synthetic data set size	5K, 7K, 15K, 38K	-

Table 5.1: Values of different query parameters used in our experiments

5.1.4 Implementation Language and Tools

The project of our experiment is implemented using C++ language and have been compiled, debugged and tested using Microsoft Visual Studio 2015 Enterprise edition with a full version student-license from DreamSpark.

5.1.5 PC configuration

We have run our experiments in three PCs of the following configurations:
1. Intel Core i5 2.9 GHz (Quad Core), 12 GB RAM
2. Intel Core i5 2.3 GHz (Quad Core), 4 GB RAM
3. AMD FX 6100 3.3 GHz (Hexa Core), 8 GB RAM
The average of these multiple runs is taken to measure and compare the performance of both the nave approach and our approach.

In Section 5.1.6, we compare the results of two approaches.

5.1.6 Comparison of Our Approach with Naive aproach

Effect of Clients movement length: Figure 5.1(a) 5.1(b)5.1(c) and 5.1(d) show processing time,server query, i/o (obstacle) and i/o(POI) respectively, for processing spatial alarm queries using naive approach and our approach. We observe that for both algorithms processing time and server query increase with the increase in length of user's movement. We also observe that in terms

of server query our approach performs almost 14times better than the naive approach. and incase of runtime our approach runs almost 3 times better than the naive approach. Incase of I/o our approach initially is higher than the naive approach but, in the average case our approach is better than the naive approach by approximately 10 times.

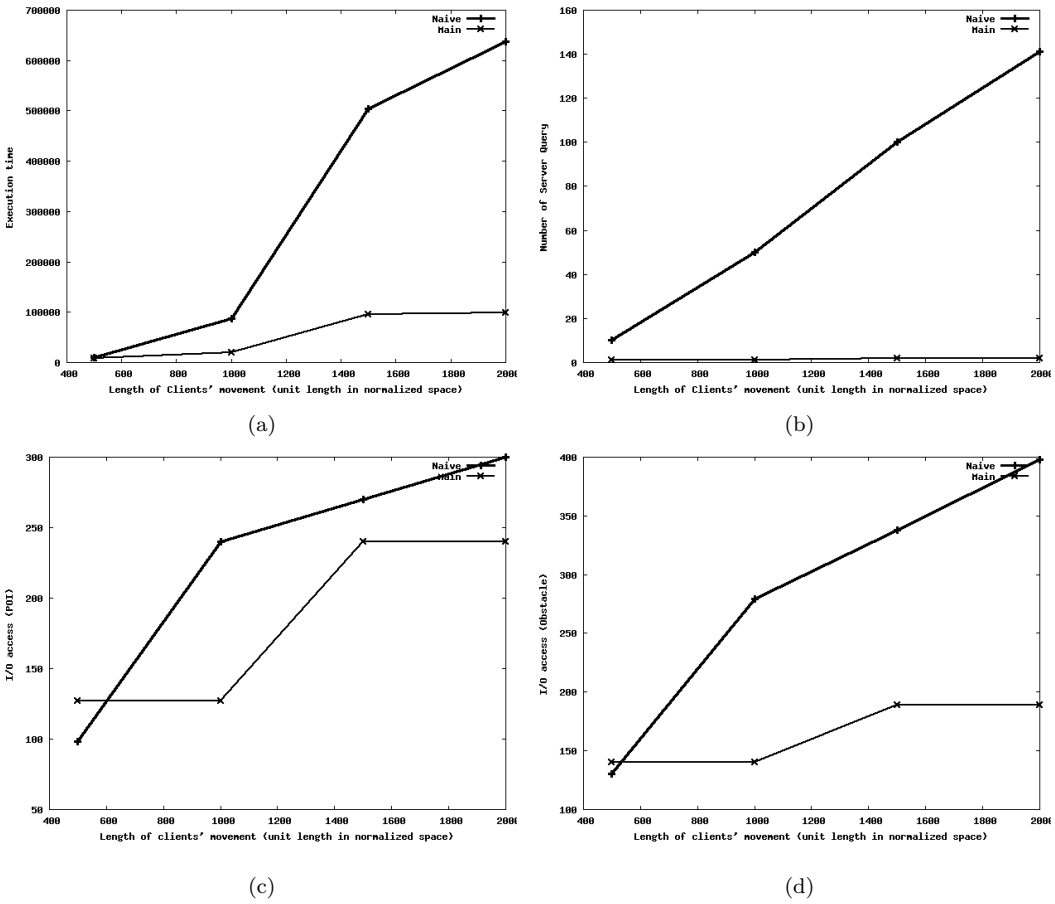

Figure 5.1: Effect of Client's movement length for Germany data (a)query processing time and (b)Server Query (c)IO-POI (d)IO-obstacle

Effect of Alarm range r***:*** 5.2(b)5.2(c) and 5.2(d) show processing time,server query, i/o (obstacle) and i/o(POI) respectively, for processing spatial alarm queries using naive approach and our approach. We observe that for both algorithms processing time and server query increase with the increase in length of alarm range. We also observe that in every alarm range our approach

outperforms the naive approach by multiple times.

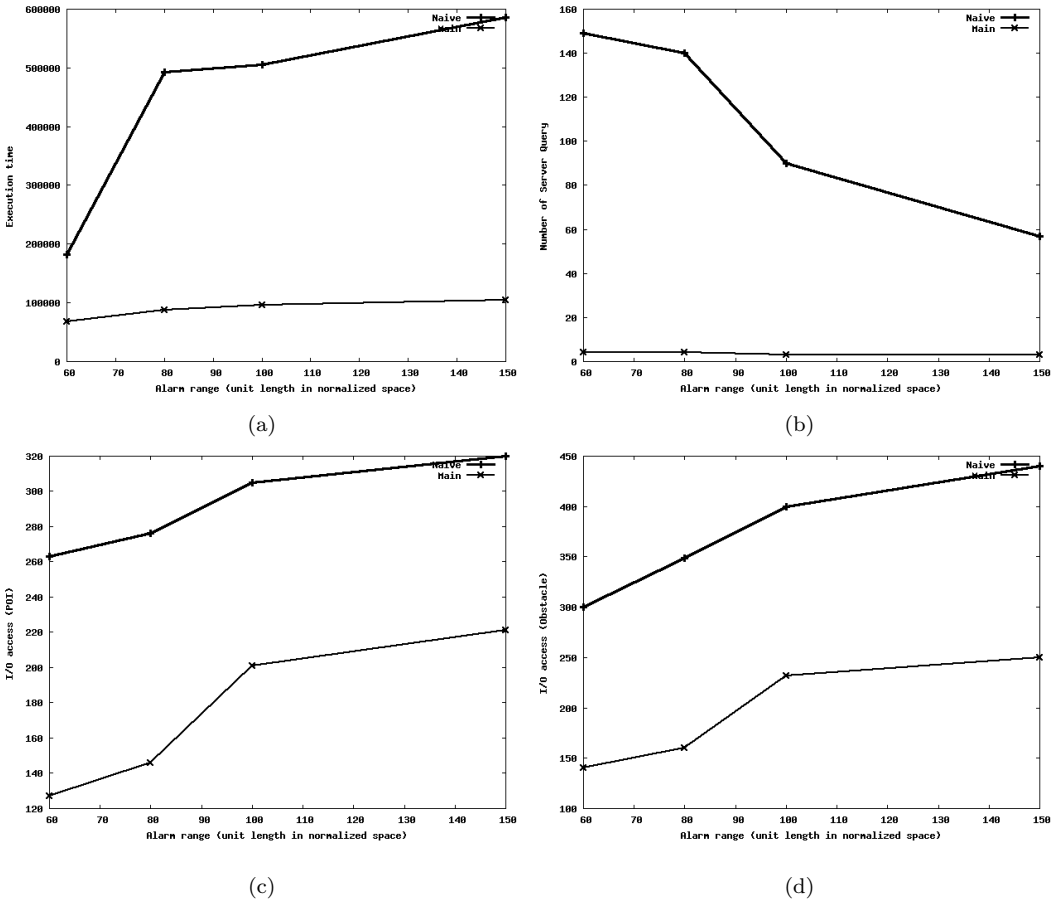

Figure 5.2: Effect of Alarm Range r for Germany data (a)query processing time ,(b)Server Query ,(c)I/O(obstacle) and (d)I/O(POI)

5.3(b)5.3(c) and 5.3(d) show processing time,server query, i/o (obstacle) and i/o(POI) respectively, for processing spatial alarm queries using naive approach and our approach. We observe that for both algorithms processing time and server query increase with the increase in the size of synthetic dataset. We use uniform distribution of real dataset to produce datasets of size 5k,7k,15k and 38k. We also observe that in every synthetic dataset size our approach outperforms the naive approach by multiple times.

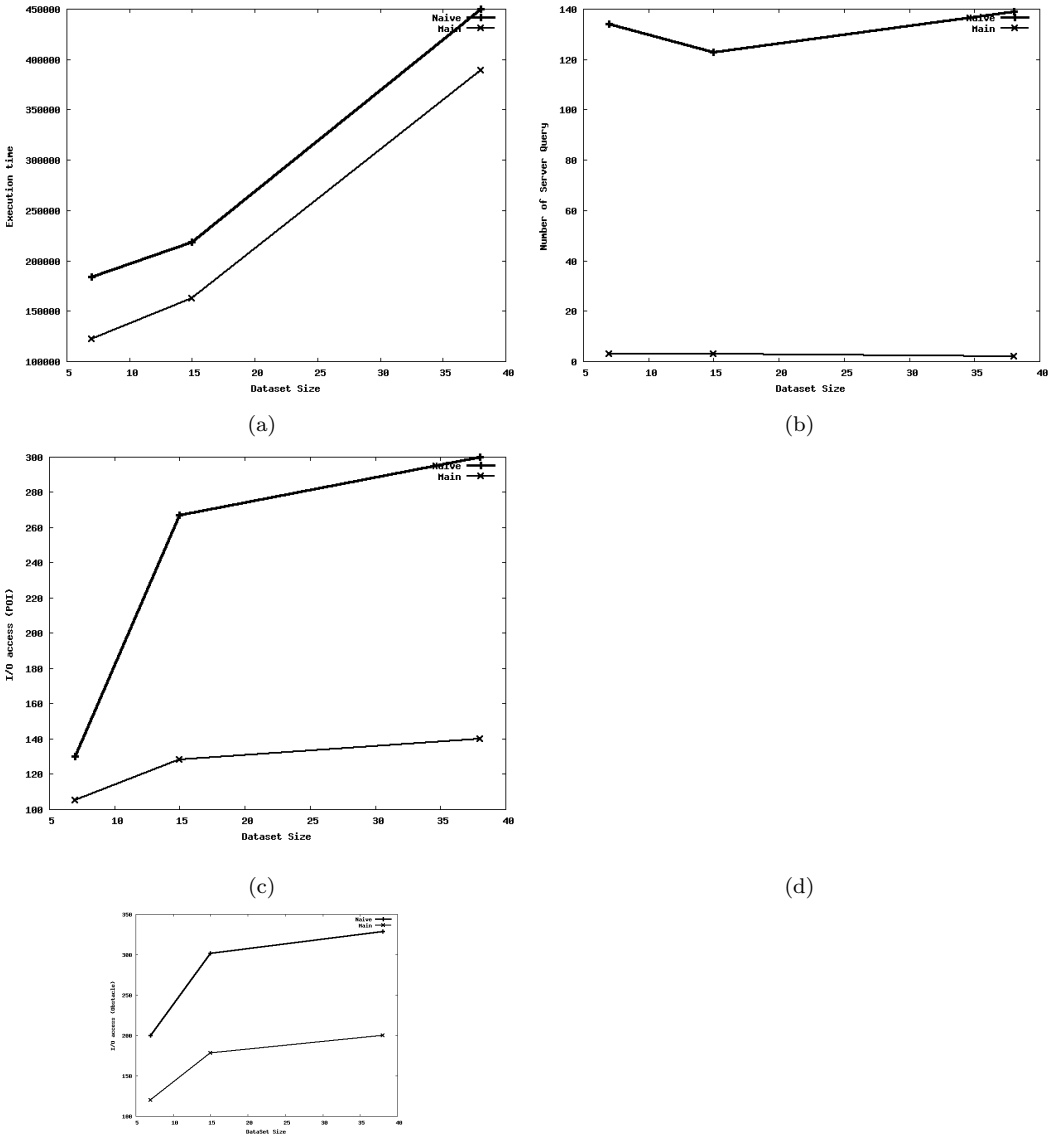

Figure 5.3: Effect of synthetic data set size for Germany data (a)query processing time ,(b)Server Query ,(c)I/O(obstacle) and (d)I/O(POI)

Chapter 6

Conclusion

In this paper we have addressed evaluation of spatial alarm in obstructed space for the first time. We have proposed two variations of our approach to incorporate both accuracy and efficient communication bandwidth.Our experimental setup provides a comparative analysis between our approach and the naive approach on different parameters. The results of the experiment conducted shows that our proposed approach is better than the naive approach in execution time, IO access and number of server queries.

6.1 Future Directions

In the future we wish to extend our thesis work in several directions.

- In future, we aim to manipulate the geometrical properties of the regions to find a larger safe region.
- We wish to find a better prediction function for approximating the clients' next direction.
- In the future we wish to provide authentication and privacy while accessing the user's location.