Abstract

Spatial Alarms are personalized Location Based service(LBS), that are triggered by a specific location of a moving user, instead of time. In this paper, we introduce an efficient algorithm to evaluate spatial alarm queries in obstructed space. Existing work in this area has focused mainly on Euclidean distance and road network models. The key idea of our approach is to compute a specific region within which the answer set of our query remains unchanged. Our aim is to reduce redundant computations in client side, while preserving the accuracy of the alarms triggered.

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 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: We define two different known regions for the POIs and the obstacles. The region containing at least 1 POI is the known region for POI.
The region circulating the POIs known region containing none or single colliding obstacles is the known region for the obstacles. The set of obstacles and POIs within this region is known to the client. 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

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

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 smartphones 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

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. In Subsections 4.2.1, 4.2.2, 4.2.3, and 4.2.4, we discuss the steps of the algorithm in detail. Finally Section **??** proposes the improvements of this algorithm to save computational cost and client-server communication overheard.

2.1 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.

2.2 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.

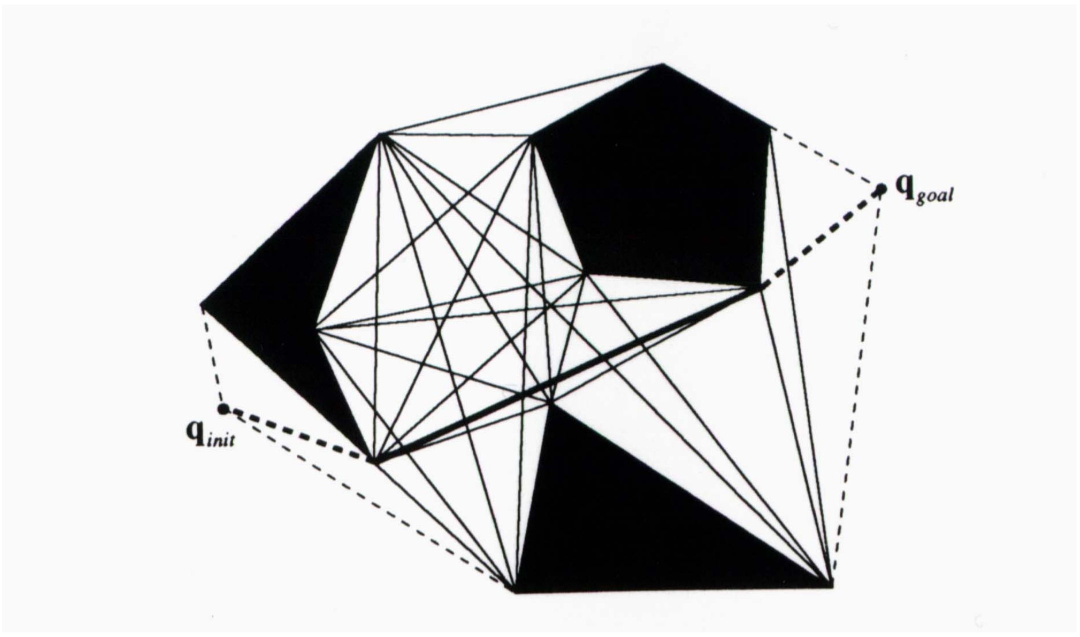


Figure 2.1: Visibility Graph

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

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

Algorithm 1: GetAlarmables(q, r)

```
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 $V_G \leftarrow$ MakeVisGraph$(P, O)$
4 return MakeAnswerSet$(q, V_G)$
```

set A consisting of the region's center q, minimum distance to be covered by the client to trigger this update procedure $kmin$, 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 $p_i \in P$ 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 3

Our Approach

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

- 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 collisions 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 Mode* and *Computation Saving Mode.*

3.0.1 Computation of the regions

The known region is bounded in one side by a parabola, whose focus is the user's location q, the equation of the parabola being $y^2 = 4ax$ 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. If the user is likely to move along a straight line, the exposure of the parabola need not be very high. But if the user is likely to move away from the major axis of the parabola, the exposure should be accordingly large. A function will be defined later on which estimates the value of m from the changes in user's direction in his previous trajectory. However,the function will ensure that the minimum value of m will be 2. This parabola is bounded on the other side by a straight line parallel to the latus rectum of the parabola. The distance of this straight line from the user's current location is dependent upon the previous history of the user's movements.Thus, we will position the bounding straight line of the known region at a distance nr from the focus. where the value of n is dependent on the user's previous trajectory. Intuitively we can see that, if the user moves fast, our known region should be larger, as the user is likely to move through the region quickly. Thus n is proportional to user's velocity v. We define a function later in the paper, which will compute the value of n.The minimum value for n is 1.

The computation of reliable region is comparatively less complex. We simply create a bounded region by subtracting r from each vertex of known regions boundary.

The safe region is computed using the nearest POI p, where $dist_E(p, q) > r$, then the safe region is the intersection of the circle centered at q with radius $dist_E(p, q)$ and the reliable region.

3.0.2 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 three parts - *client-initialization from server side*, *alarm-configuration* and *update on any minimal amount of location change*. The Algorithms 1, 2, 3 show the algorithmic-steps for these three parts respectively.

The input to the client-initialization algorithm (Algorithm 1) is the current location of the client q, query radius r, current velocity of the client v, path history as a set of straight lines $S = (m_1, c_1, l_1)$, (m_2, c_2, l_2), (m_3, c_3, l_3), ... (m_n, c_n, l_n) and the last answer set $A_{prev} = q$, u, m_p, m_o, n, where u is the vertex of the . This algorithm makes the frequent queries more efficient and

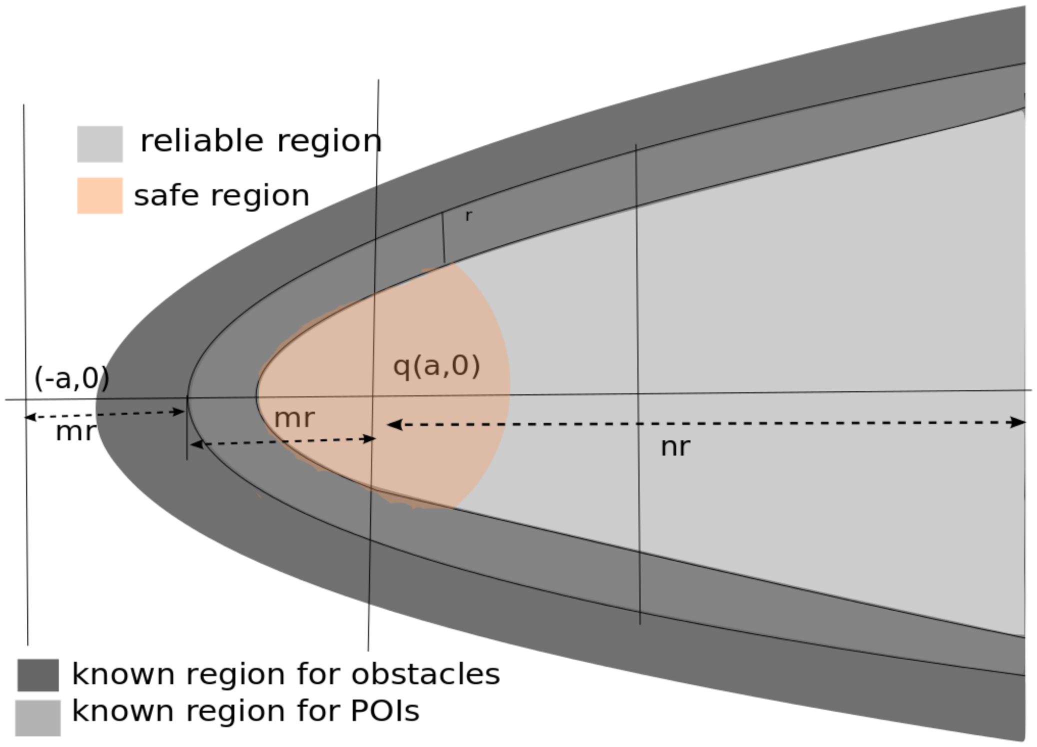


Figure 3.1: Different regions

accurate and needs much less server communication.

In the following algorithms, 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 GETVERTEX(q, s_{dir}) returns the vertex point u of the parabola having focus at q and the axis along the vector s_{dir}.

The function VARIANCEOFPATH(S) returns a real number within the range $[2, \infty)$ 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.

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

GETCOLLISIONPOINT(O, q, m_o, r, n_o) checks any collision of any of the obstales from the set O with the parabola $y^2 = 4*(m_o*r)*x$ and also the bounding straight line at distance (n_o*r) from the focus q and perpendicular to the axis of the parabola. Then it returns an all negative point if there is no collision, otherwise returns a positive point in the space. The ISPOSITIVEPOINT(p) reutrns true for the point p to be positive, otherwise returns false. Again, the function POINTONSTRAIGHTLINE (p, n, r, q) returns true if the point p is on the straight line at distance $(n*r)$ 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 output of the Algorithm 1 is an answer set A which consists of the radius of the known regions of POIs and obstacles (r_{kp} and r_{ko} respectively), the set O of all obstacles within radius r_{ko} and the set of all POIs within radius r_{kp} centring q. Since it is a bandwidth saving mode, the visibility graph is not sent to the client as long as it can be computed using the existing data in the client side.

The input to the Algorithm 2 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.

In this algorithm, the visibility graph is computed using the answer sets P and O got as return of the 1 from the server side. The input to the algorithm 3 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 3: QUERYALARMABLES(q, r, v, S, A_{prev})

```
Input  : Query point $q$, query radius $r$, velocity $v$, path history as a set
         of straight lines $S$, last answer set $A_{prev}$
Output: The answer set, $A \leftarrow \{q, u, m_p, n_p, m_o, n_o, P, O\}$
1  $s_{dir} \leftarrow$ PREDICTDIRECTION($S$)
2  $u \leftarrow$ GETVERTEX($q, s_{dir}$)
3  $m_p \leftarrow$ VARIANCEOFPATH($S$)
4  $n_p \leftarrow ln(e + v)$
5  $O \leftarrow$ GETOBSTACLESET($q, m_p * r, n_p$)
6  $V_G \leftarrow$ MAKEVISGRAPH($P, O$)
7  $m_o \leftarrow m_p$
8  $n_o \leftarrow n_p$
9  $p_{col} \leftarrow$ GETCOLLISIONPOINT($O, q, m_o, r, n_o$)
10 while ISPOSITIVEPOINT($p_{col}$) do
11     $O' \leftarrow \emptyset$
12     $P' \leftarrow \emptyset$
13     $b \leftarrow$ POINTONSTRAIGHTLINE($p_{col}, n_o, r, q$) if $b$ then
14         $O' \leftarrow$ GETOBSTACLESET($q, u, m_o * r, n_o + 1$)
15         $P' \leftarrow$ CHECKNEWPOI($q, u, m_o * r, n_o + 1$)
16         $n_o \leftarrow (n_o + 1)$
17     else
18         $O' \leftarrow$ GETOBSTACLESET($q, u, (m_o + 1) * r, n_o$)
19         $P' \leftarrow$ CHECKNEWPOI($q, u, (m_o + 1) * r, n_o$)
20         $m_o \leftarrow (m_o + 1)$
21     if $|P'| > |P|$ then
22         $P \leftarrow P'$
23         if $b$ then
24             $m_p \leftarrow (m_p + 1)$
25         else
26             $n_p \leftarrow (n_p + 1)$
27         $V_G \leftarrow$ ATTACHTOVISGRAPH($V_G, P' - P, O' - O$)
28     $O \leftarrow O'$
29     $p_{col} \leftarrow$ GETCOLLISIONPOINT($O, q, m_o, r, n_o$)
30 return $A \leftarrow$ MAKEANSWERSET($q, u, m_p, n_p, m_o, n_o, P, O$)
```

Algorithm 4: ConfigUpdate(q, A)

```
    Input   : Query point $q$, answer set $A$
1   $P \leftarrow GetPOIs(V_G)$
2   $O \leftarrow GetObstacles(V_G)$
3   foreach $p_i \in P$ do
4       $p_i.d_O \leftarrow dist_O(q, p_i)$
5       if $p_i.d_O r$ then
6           AlarmUser($p_i$)
7       else if $p_i.d_E < r_{safe}$ and isPathInside($q, p_i, r_{safe}, V_G$) then
8           $r_{safe} \leftarrow p_i.d_E$
9   return $r_{safe}$
```

Algorithm 5: UpdateOnLocChange(q, r_{safe}, r_{rel}, A)

```
    Input   : $q, r_{safe}, A$
    Output: $d_u$
1   if isOutsideReliableregion(q, A) then
2       $A \leftarrow$ QueryAlarmables($q, d_q$)
3       $r_{safe} \leftarrow$ ConfigAlarm($q, A$)
4   if $d_q > r_{safe}$ then
5       $r_{safe} \leftarrow$ ConfigAlarm($q, A$)
6   return $d_u \leftarrow r_{safe}$
```

3.0.3 Computational Cost Saving Mode

The algorithms run in this mode almost similarly as in the "*Bandwidth Saving Mode*" with all the 3 described parts - *client-initialization from server side*, *alarm-configuration* and *update on any minimal amount of location change* as demonstrated in the Algorithms **??**, 2, 3.

However, the main difference with the previously descried mode from this mode is - the Algorithm 1 returning the computed V_G as another element of the answer set A from the server side and the algorithm 2 not reconstructing this V_G in the client side during running the Algorithm 2, whereas the Algorithm 3 remains all the same.

Therefore, an $O(n^2)$ 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 1.

Chapter 4

Experimental Study

4.1 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

4.1.1 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.

4.1.2 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.

4.1.3 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 4.1 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 (km)	40,100,300,400	40
Alarm Range r	40, 60, 100, 150, 200	150
Synthetic data set size	5K, 10K, 15K, 20K	-

Table 4.1: Values of different query parameters used in our experiments

4.1.4 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.

4.1.5 PC configuration

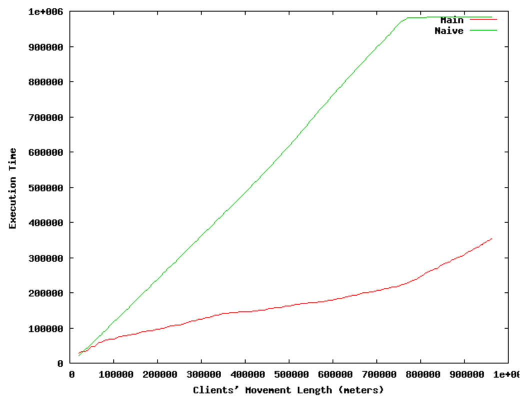
We have run our experiments in three PCs of the following configurations:
1. Intel Core i5 2.9 GHz (Quad Core), 12 GB RAM
2. Intel Core i5 2.3 GHz (Quad Core), 4 GB RAM
3. AMD FX 6100 3.3 GHz (Hexa Core), 8 GB RAM

The average of these multiple runs is taken to measure and compare the performance of both the nave approach and our approach.

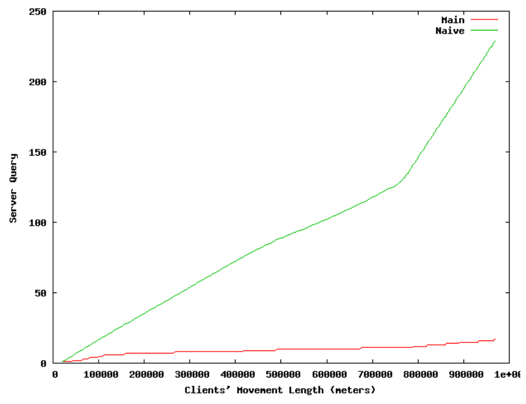
In Section 4.1.6, we compare the results of two approaches.

4.1.6 Comparison of Our Approach with Naive aproach

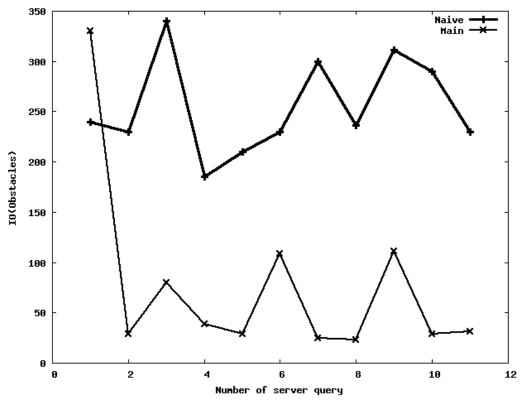
Effect of Clients movement length: Figure 4.1(a) 4.1(b)4.1(c) and 4.1(d) show processing time,server query, i/o (obstacle) and i/o(POI) respectively, for processing spatial alarm queries using naive approach and our approach. We observe that for both algorithms processing time and server query increase 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.



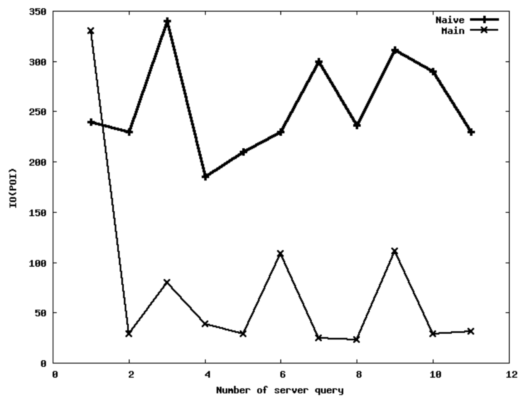
(a)



(b)



(c)



(d)

Figure 4.1: Effect of Client's movement length for Germany data (a)query processing time and (b)Server Query (c)IO-obstacles (d)IO-POI

Chapter 5

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.

5.1 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.