Spatial Alarms in the Obstructed Space*

Md. Touhiduzzaman
Department of Computer Science
Bangladesh University of Engineering and Technology
Dhaka,Bangladesh
tz08128@gmail.com

Sezana Fahmida
Department of Computer Science
Bangladesh University of Engineering and Technology
Dhaka,Bangladesh
sezanafahmida@gmail.com

Tanzima Hashem
Department of Computer Science
Bangladesh University of Engineering and Technology
Dhaka,Bangladesh
tanzimahashem@gmail.com

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.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

General Terms

Theory

Keywords

Spatial Alarm, Obstructed Space

1. INTRODUCTION

The widespread use of smart phones has led to the proliferation of LBSs. 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. Spatial Alarms are an important class of LBS,that are triggered by a specific location of a moving user, irrespective of time. "Remind me if I'm within 100 meters of a pharmacy" is a possible example of a spatial alarm.

Existing work in this area has focused mainly on Euclidean distance and Road Network models [1],[4],[2]. However, Spatial alarm evaluation in the obstructed space is different than road network or Euclidean space as it considers the obstacles in the path to the location of alarm. It is better approximated by a pedestrian scenario while road networks are approximated by vehicle scenarios. A pedestrian's path is not limited by roads. However, a pedestrian is obstructed by various obstacles such as buildings or trees. Thus while calculating the distance from alarms, we have to consider the obstructed distance[**?**]. In this paper we propose a unique approach to evaluate spatial alarms in obstructed space. To the best of our knowledge this query has not been addressed in any existing research work.

Spatial alarms are based on a predefined location with a given alarming zone radius thus applying the techniques that are used in answering spatial range queries is both inefficient and wasteful as, in spatial range query continuous re-evaluation of user's location is needed in case of a mobile user. However,in spatial alarm, the user's location is not relevant at all times. 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. Another challenge in spatial alarm processing is the communication cost between the server and the client.

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. 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.

*(For use with SIG-ALTERNATE.CLS. Supported by ACM.

2. 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 [1] 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 [5] spatial alarms 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 [6] 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 $dist_o(p_i, q) < r$

3. 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, correctness and scalability. Correctness refers to the quality that guaranties no alarms are missed. And scalability is the feature that measures the number of POI's the system can adapt to. In this paper, We propose a novel approach to evaluate spatial alarms in obstructed space which ensures both high accuracy and high scalability.

In this paper, we adopt the concepts of three different types of regions: *Known Region* [**?**], *Reliable Region* [**?**] and *Safe Region* [**?**]. The modified definition of these regions according to our context are as the followings:

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. We will denote the radius of the known region of the POIs as r_{kp} and that of the obstacles as r_{ko}.

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. Given the radius of the reliable region as r_{rel}, the user's previous location as P_1 and the current location as P_2, if $(P_1 - P_2) < r_{rel}$, then by this definition no further queries to the server has to be done to compute a consistent set of answers.

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.

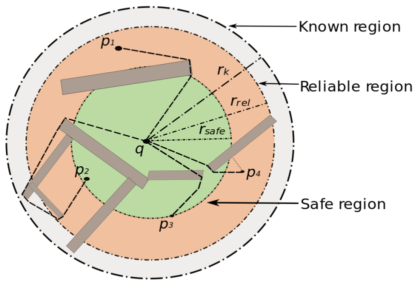

Figure 1: Known, reliable and safe regions

Table 1: Symbol Table

Symbols	Meaning
P	Set of POIs
O	Set of Obstacles
r	Alarming range
dist$_E$(p,q)	Euclidean distance between point p and q
dist$_O$(p,q)	Obstructed Distance between point p and q
r_{safe}	radius of safe region
r_{rel}	radius of reliable region
r_{kp}	radius of known region for POIs
r_{ko}	radius of known region for obstacles
p_i	any $i^{th} POI$

4. RELATED WORK

4.1 Spatial Alarms

Extensive research has been performed and various effective algorithms have been proposed [4],[5],[1] 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 on the other hand in road networks navigation is limited along predefined roads. [4]proposes a solution to the spatial alarm problem for moving users on road network. They introduce road network-based spatial alarms using segment length-based and travel time-based road network distances. Our approach aims to provide a solution to the same problem, depicted in an obstructed space[?] scenario. Again, their solution incorporates the concept of hibernation time, a time during which no processing takes place in the mobile client or the processing engine comprehensive research, where in our approach we propose the novel concept of safe region. Comprehensive research has been conducted in [2] to make spatial alarm evaluation energy-efficient and effective in road networks. Though to the best of our knowledge, no efficient algorithm has yet been devised to process spatial alarms in obstructed space. [?] provides an efficient indexing structure for the processing of spatial alarms called the Mondrian tree. However, in our approach we have used the conventional R-tree structure.

4.2 Obstructed Space

Obstructed space considers the shortest distance between two points in the presence of obstacles.[?] Various spatial range query algorithms have been presented in recent times [7],[3],[6] such as nearest neighbor and group nearest neighbor in obstructed space.[6] provides an efficient approach to find the aggregate obstructed distance along with processing the group nearest neighbor query in obstructed space.[7] provides efficient algorithms for range search, nearest neighbors, e-distance joins and closest pairs, in obstructed space,considering that both data objects and obstacles are indexed by R-trees. Again, in [3] efficient algorithms have been computed for the reverse nearest neighbor query. [?] proposes a safe region based approach to comptuing moving k-nearest neighbour queries in obstructed space.Our query is quite dissimilar to the queries mentioned above in the sense that, while computing spatial alarms, we have to return all POIs which are within the alarming zones radius of the client instead of the nearest or a group of nearest POIs. [?] has computed a known region for obstacles for convenience in computing the safe region.In our approach we have also included a known region for POIs. Again, the computation technique for known region of obstacles or obstructed known region as referred in [?] is quite different from ours. Our approach is to incrementally increase the radius of the known region until all the POI's are visible, where in their paper, an obstructed known region is the region that consists of points that are of equal or less distance from the user than a datapoint. Their safe region computation technique although quite efficient in computing knn queries, is not optimal in evaluating spatial alarms. However, to the best of our knowledge no research work has yet been published on the topic of spatial alarms in obstructed space.

5. NAIVE APPROACHES

In all of our naive and even main approaches, we have assumed that the full system is divided into a client-server architecture and the POIs as well as the obstacles are saved in an R-tree at the server side. In this paper 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-phones or PDA.

To compare the efficiency of our approach, we are about to present two straightforward solutions for processing spatial-alarms in an obstructed space - sequential POI processing on regular basis and region-divided alarm (POI) processing.

Both of the naive approaches are explained in the following subsections.

5.1 Sequential Processing on Regular Basis

In this naive approach, all nearest POIs are retrieved from the server to populate the answer set A with the set of POIs (P), set of obstacles (O) and the visibility-graph V_G and these are frequently checked to be crossed later in the client side.

In the following algorithms, the $\textsc{GetAllPOI}(q, r)$ function populates the set P of POIs within the radius r centring the query point (the client's current location) q. Similarly, $\textsc{GetObstacleSet}(q, r)$ function returns the set of all obstacles within the radius r centring q.

$\textsc{MakeVisGraph}(P, O)$ 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.

$\textsc{MakeAnswerSet}(q, d_u, P, O, V_G)$ function returns a heterogeneous data-model consisting of its all parameters to be used by the caller client-side program.

Algorithm 1: $\textsc{InitByServer}(q,\ r)$

Input : Query point q and the search radius r
Output: Answer set A
$P \leftarrow \textsc{GetAllPOI}(q, r)$
$O \leftarrow \textsc{GetObstacleSet}(q, r)$
$V_G \leftarrow \textsc{MakeVisGraph}(P, O)$
$u_{min} \leftarrow \infty$
foreach $p_i \in P$ **do**
  $u_{min} \leftarrow min(u_{min},\ p_i.u)$
$d_u \leftarrow (r_{max} - u_{min})$
return $A \leftarrow \textsc{MakeAnswerSet}(q, d_u, P, O, V_G)$

The following method is triggered on a minimum 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 $\textsc{AlarmUser}(p_i)$ 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 changed client-location q' and the answer set A consisting of the regions 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.

In this naive-approach, the visibility-graph is constructed more times than necessary to hold accuracy, each of which constructions requires $O(n^2)$ [5], where $n =$ the number of edges of the obstacles. A huge overhead is also sufficed to

Algorithm 2: UPDATECLIENT(q, A)

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

foreach $p_i \in P$ **do**
  if $dist_O(q, p_i, V_G) > p_i.u$ **then**
    ALARMUSER(p_i)
if $dist_E(A.q, q) > A.d_u$ **then**
  INITBYSERVER$(q, 100)$

make P and O sets using such procedure. Again, the Algorithm 8 can also be run much less time than in this approach, which is improvised in the later approaches.

5.2 Region-based Alarm Processing

This naive approach is a region-based modified straight forward approach which searches for a new POI inside the known region as soon as the client changes it's position. This naive algorithm works in three parts - computing the known and reliable regions using the retrieved POIs and obstacles, checking the client's status to give alarms using the computed visibility graph and checking the region crossings to recompute the answer set on any minimal location-change of the client. The Algorithms 3,4,5 show the respective parts.

Instead of the very naive GETALLPOI(q, r) function used in the first naive approach, a modified function CHECKNEWPOI$(q, r, q_{prev}, r_{prev})$ is used in our following approaches to retrieve all excluded POIs of the set P already in the client's side, where $\forall p_i \in P$ $dist_E(q_{prev}, p_i) \leqslant r_{prev}$. So, the new set of POIs P' got from this method will be such as $\forall p_i \in P'$ $dist_E(q, p_i) \leqslant r$.

Similarly, the GETOBSTACLESET(q, r) is modified to GETOBSTACLESET$(q, r_k, q_{prev}, r_{prev})$.

The input to the Algorithm 3 is the current location of the user q and the increment delta r_d, which is the amount by which the region will be expanded. The output of the algorithm is an answer set A which consists of the set of all obstacles O and the set of all POIs P along with the respective visibility graph V_G within the known-region radius r_k centring q.

Algorithm 3: INITREGIONSBYSERVER(q, r_d)

Input : Query point q, increment delta r_d
Output: The answer set, $A \leftarrow \{r_k, P, O\}$

while $|P| < 1$ **do**
  $r_k \leftarrow (r_k + r_d)$
  $P \leftarrow$ CHECKNEWPOI$(q, r_k, q_{prev}, r_{prev})$
$O \leftarrow$ GETOBSTACLESET$(q, r_k, q_{prev}, r_{prev})$
$V_G \leftarrow$ MAKEVISGRAPH(P, O)
return $A \leftarrow$ MAKEANSWERSET(q, r_k, P, O, V_G)

In the Algorithm 4, the function ISANYPOIUNREACHABLE(V_G) returns true if any POI inside the visibility graph V_G cannot be reached along any path inside V_G.

The input to the Algorithm 4 is the current location of the client q, the answer set from the Algorithm 3 and the increment amount r_d by which the known region wil be more expanded if needed. This algorithm is also responsible for triggering an alarm to the client, if s/he is within the alarming distance $p_i.u$ of any POI $p_i \in A.P$.

Algorithm 4: SAFEREGIONCALC(q, A, r_d)

Input : Query point q, latest answer set A
Output: r_{safe}

if ISANYPOIUNREACHABLE(V_G) **then**
  $A \leftarrow$ INITREGIONBASE(q, r_d)
  return SAFEREGIONCALC(q, A)
$D_{min} \leftarrow \infty, u_{max} \leftarrow 0$
foreach $p_i \in A.P$ **do**
  $p_i.u \leftarrow$ GETPOIRANGE(p_i)
  $D_i \leftarrow dist_O(q, p_i, V_G)$
  if $D_i \leqslant p_i.u$ **then**
    ALARMUSER(p_i)
  else
    $D_{min} \leftarrow min(D_{min}, D_i)$
    $u_{max} \leftarrow max(u_{max}, p_i.u)$
$r_{rel} \leftarrow (r_k - D_{min})$
return $r_{safe} \leftarrow (D_{min} - u_{max})$

The input to the Algorithm 5 is the current and the previous location of the client (q and q_{prev}), the radius of the safe region(r_{safe}), the reliable region(r_{rel}) and the known region(r_k) and finally the latest computed answer set A. The Algorithm 5 returns the minimum distance d_u to trigger this algorithm the next time.

Algorithm 5: OnLocationChange$(q, q_{prev}, r_{safe}, r_{rel}, r_k)$

Input : $q, q_{prev}, r_{safe}, r_{rel}, r_k, A$
Output: d_u

$d_q \leftarrow dist_E(q, q_{prev})$
if $d_q > r_{rel}$ **then**
  $A \leftarrow$ INITREGIONBASE(q, d_q)
  $r_{safe} \leftarrow$ SAFEREGIONCALC(q, A, d_q)
else if $q_d > r_{safe}$ **then**
  $r_{safe} \leftarrow$ SAFEREGIONCALC(q, A, d_q)
return $d_u \leftarrow r_{safe}$

In this approach, more than one query for the known region computation has to be done frequently to the server after computing the visibility graph and also if any unreachable POI is found out in the Algorithm 4 - which seems very much inefficient.

Moreover, the safe region is in its minimum size in this approach, which requires more computation in the client side and thus gets the approach less efficient.

These problems are solved in the following final approach.

6. 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*.

6.1 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 6, 7, 8 show the algorithmic-steps for these three parts respectively.

The input to the client-initialization algorithm (Algorithm 6) is the current location of the client q and the incremental radius r_d by which the radius of the searchable region will expand. This algorithm improvises the single known region concept of the second naive approach described above into two different known regions for POIs and obstacles, which makes the frequent queries more efficient and accurate and needs much less server communication.

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 6 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.

Algorithm 6: ClientInitByServer(q, r_d)

Input : Query point q, increment delta r_d
Output: The answer set, $A \leftarrow \{r_{kp}, r_{ko}, P, O\}$
while $|P| < 1$ **do**
    $r_{kp} \leftarrow (r_{kp} + r_d)$
    $P \leftarrow$ CheckNewPOI$(q, r_{kp}, q, 0)$
$r_{ko} \leftarrow r_{kp}$
$O \leftarrow$ GetObstacleSet$(q, r_{ko}, q_{prev}, 0)$
$V_G \leftarrow$ MakeVisGraph(P, O)
while isAnyPoiUnreachable(V_G) **do**
    $O' \leftarrow$ GetObstacleSet$(q, r_{ko}, q, r_{ko} + r_d)$
    $P' \leftarrow$ CheckNewPOI$(q, r_{ko}, q, r_{ko} + r_d)$
    $r_{ko} \leftarrow (r_{ko} + r_d)$
    if $|P'| > |P|$ **then**
        $P \leftarrow P'$
        $r_{kp} \leftarrow r_{ko}$
        $V_G \leftarrow$ AttachToVisGraph$(V_G, P' - P, O' - O)$
    $O \leftarrow O'$
return $A \leftarrow$ MakeAnswerSet(r_{kp}, r_{ko}, P, O)

In the Algorithm 7, isPathInside(q, p_i, r_{safe}, V_G) returns true if the POI p_i is inside the radius r_{safe} centering q along with all of its connecting edges in V_G, otherwise returns false. The input to the Algorithm 7 is the current location of the client q and the answer set got from the Algorithm 6. This algorithm is also responsible for triggering an alarm to the client if s/he is within the alarming distance of any POI. Recall that, here $\forall p_i \in P$, $p_i.u$ is the alarming radius, $p_i.d_E$ is the Euclidean distance from the center of the regions and $p_i.d_O$ is the obstructed distance in the visibility graph V_G.

Algorithm 7: ConfigUpdate(q, A)

Input : Query point q, answer set A
Output: r_{safe}
$V_G \leftarrow$ MakeVisGraph(P, O)
$r_{safe} \leftarrow \infty, u_{max} \leftarrow 0$
foreach $p_i \in P$ **do**
    $p_i.u \leftarrow$ GetPoiRange(p_i)
    $r_{safe} \leftarrow min(r_{safe}, p_i.d_O - p_i.u)$
    $u_{max} \leftarrow max(u_{max}, p_i.u)$
$r_{rel} \leftarrow (r_{ko} - u_{max})$
foreach $p_i \in P$ **do**
    if $p_i.d_O \leqslant p_i.u$ **then**
        AlarmUser(p_i)
    else if $p_i.d_E < r_{safe}$ *and* isPathInside(q, p_i, r_{safe}, V_G) **then**
        $r_{safe} \leftarrow p_i.d_E$
return r_{safe}

In this algorithm, the visibility graph is computed using the answer sets P and O got as return of the 6 from the server side. The most critical part of this algorithm is to calculate the safe region. This is explained more with a suitable example depicted in the figure 4 and described later in this section.

The input to the algorithm 8 is the current and the previous location of the client (q and q_{prev}), the radius of the safe region(r_{safe}) and the reliable region(r_{rel}) 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 8: UpdateOnLocChange$(q, r_{safe}, r_{rel}, A)$

Input : $q, q', r_{safe}, r_{rel}, A$
Output: d_u
$d_q \leftarrow dist_E(A.q, q)$
if $d_q > r_{rel}$ **then**
    $A \leftarrow$ ClientInitByServer(q, d_q)
    $r_{safe} \leftarrow$ ConfigAlarm(q, A)
if $d_q > r_{safe}$ **then**
    $r_{safe} \leftarrow$ ConfigAlarm(q, A)
return $d_u \leftarrow r_{safe}$

6.2 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*

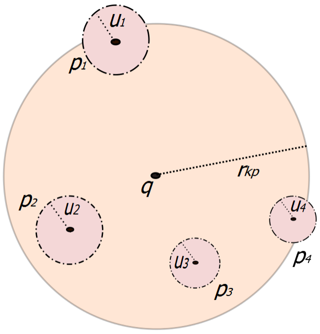

Figure 2: Retrieved POIs within r_{kp} centring q

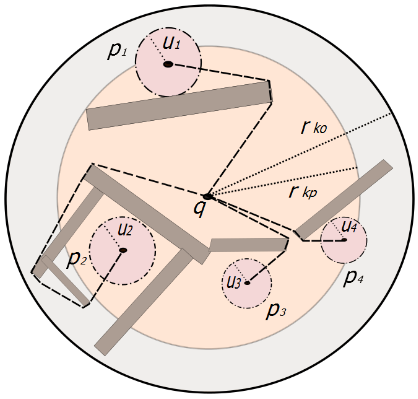

Figure 3: Generated V_G

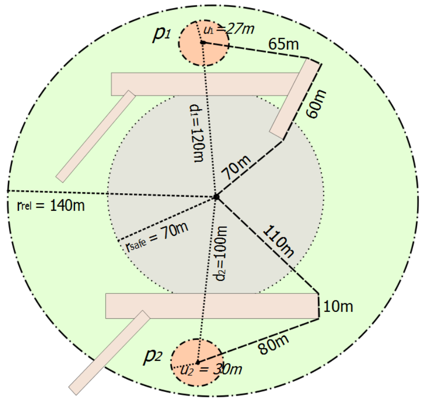

Figure 4: Safe Region Calculation

and *update on any minimal amount of location change* as demonstrated in the Algorithms 6, 7, 8.

However, the main difference with the previously descried mode from this mode is - the Algorithm 6 returning the computed V_G as another element of the answer set A from the server side and the algorithm 7 not reconstructing this V_G in the client side during running the Algorithm 7, whereas the Algorithm 8 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 6.

The above described approach can be depicted with some suitable examples as presented in the Figures 2, 3 and 4.

In figure 2 an example scenario is put as if 3 POIs p_1, p_2, and p_3 are retrieved and in the figure 3 the same case is shown as if the algorithm 6 has returned the constructed visibility-graph V_G along with the set of POIs P, obstacles O and the radius of the POIs' known region r_{kp} and that of the obstacles' known region r_{ko}.

In the Figure 4, a critical case for the Algorithm 7 is demonstrated regarding the calculation of the safe region.

During the for-loop at line no. 4 of the algorithm 7, the safe-region's radius is calculated to be,
$r_{safe} = min((65+60+70)-27, (110+10+80)-30) = 168m$, whereas the safe-region is about to include both the POIs along with their whole path in the visibility graph V_G from the center q and also without any of them within their alarming zone. In this case as per the definition **??** , no calculation would be done to alarm the client even if s/he enters within any of the POI's alarming zone. To narrow down this false safe-region for the sake of accuracy, the condition of line no. 9 inside the foreach-loop gets true and the radius is modified as $r_{safe} = min(168, 120-27) = 93m$ for p_1 and in the second and final iteration for p_2, $r_{safe} = min(93, 100-30) = 70m$.

6.3 Proof of Correctness

The following proved facts bear the proof of accuracy and completeness of our final algorithms 6, 7 and 8.
In the algorithm 6, the fact that the incremental-search will find at least 1 POI and stop the increment to get a fixed r_{kp} within finite time follows from the incremental search algorithm given the fact that the POI data-set is not empty. One more fact is to be proved to guarantee the accuracy and completeness of the algorithm 6 as - the while loop at line 7 runs for a finite amount of time.

PROOF. If there's no unreachable POI or no/single collision between any obstacle and the perimeter of the POIs' known region, then the loop will terminate immediately.
If there is any totally unreachable POI, it must be surrounded by a series of obstacles, which will certainly cause no collision (in case that all obstacles are already inside the known region) or more than one collision (in case that some parts of the series of the obstacles are inside the known region) with the perimeter of the known region. So, the second clause will be false and the loop will terminate.
Finally, if there is any unreachable POI which can be reached by retrieving an extended set of obstacles, then it will be done and checked inside the loop and then loop will finish its purpose within finite time. □

No computation is needed to accurately give alarm while the user is inside the safe region.

PROOF. *Case 1*: When the path to a POI is a straight line: In this case the claim is trivial to prove. We take $min(D_i - U_i)$ as the radius of safe region. Suppose there is a POI P with alarming distance U. The radius of the safe region is r. and the users current position is p'. Suppose for contradiction an alarm should be triggered to the user for P in his current position p'. Then, $|p - p'| > (D - U)$. But as the user is within the safe region, $|p - p'| < r$. But that mean, $r > |D - U|$ which is a contradiction because the algorithm 7 chooses the minimum between all $(D_i - U_i)$.
Case 2: When the path to a POI is not a straight line: In this case there is an obstacle in the path to the POI. There can be two cases, a. the safe region contains the full path to the obstructed POI b. The safe region does not contain the full path to the obstructed POI. In case a, the algorithm 7 computes the minimum among the Euclidean distances of the POIs. As we know from the Euclidean lower bound property that the $dist_O(a, b) \geqslant dist_E(a, b)$, the proof follows from case 1. The safe region's radius will never over-assume

the distance to the POI as it is considering the Euclidean distance. In case b, the algorithm 7 chooses the safe-region radius with the assumption that as the POI's full path is not the safe region, even if the user get's close to the POI in Euclidean Distance, Obstructed distance will always be higher.(Euclidean Lower Bound) □

No query to the server has to be done to correctly give any alarm while the user is inside the reliable region
Recall from algorithms 6 and 7 that the minimum alarming distance among all the available POIs for the user is returned as U_{min}, which is used to reduce the POIs' known region's radius to the reliable region's radius as $r_{rel} = r_{ko} - U_{max}$.

PROOF. If the safe region is well inside the safe region, then this proof follows the 1st fact. The 3 procedures run simultaneously to give accurate alarm for the POIs inside the known region and so inside the reliable as well as the safe region. The proof is needed for any POI outside both the known regions.

Let there be a POI outside both the known regions for which no alarm is triggered when the user gets inside its alarming distance U_i. But meanwhile, the user must cross the reliable region because $r_{ko} - r_{rel} = U_{max} > U_i$ So according to algorithm 8, algorithm 6 and 7 are re-run and the assumed POI must come inside the newly computed known regions and its alarm will be given accurately. Hence, there is a contradiction. Therefore, there is no POI outside the reliable region which may miss its alarm. So, the statement is proved. □

The update procedure is run timely to re-calculate the answer set.

PROOF. This claim follows trivially from the proof of the fact that - no computation is needed to accurately give alarm while the user is inside the safe region. □

7. EXPERIMENTS

8. CONCLUSIONS

This paragraph will end the body of this sample document. Remember that you might still have Acknowledgments or Appendices; brief samples of these follow. There is still the Bibliography to deal with; and we will make a disclaimer about that here: with the exception of the reference to the LaTeX book, the citations in this paper are to articles which have nothing to do with the present subject and are used as examples only.

9. ACKNOWLEDGMENTS

This section is optional; it is a location for you to acknowledge grants, funding, editing assistance and what have you. In the present case, for example, the authors would like to thank Gerald Murray of ACM for his help in codifying this *Author's Guide* and the **.cls** and **.tex** files that it describes.

10. REFERENCES

[1] Bhuvan Bamba, , Arun Iyengar, and Philip S. Yu. Distributed processing of spatial alarms: A safe region-based approach. In *29th IEEE International Conference on Distributed Computing Systems (ICDCS 2009)*

[2] Myungcheol Doo, Ling Liu, Nitya Narasimhan, and Venu Vasudevan. Efficient indexing structure for scalable processing of spatial alarms. In *18th ACM SIGSPATIAL International Symposium on Advances in Geographic Information Systems, ACM-GIS*

[3] Yunjun Gao, Jiacheng Yang, Gang Chen, Baihua Zheng, and Chun Chen. On efficient obstructed reverse nearest neighbor query processing. In *19th Ling LiuACM SIGSPATIAL International Symposium on Advances in Geographic Information Systems, ACM-GIS 2011, 2011, Chicago, USA*, pages 191–200. ACM, 2011.

[4] Kisung Lee, Emre Yigitoglu, Ling Liu, Binh Han, Balaji Palanisamy, and Calton Pu. Roadalarm: A spatial alarm system on road networks. In *29th IEEE International Conference on Data Engineering ICDE, Brisbane, Australia, 2013*, IEEE Computer Society, 2013.

[5] Anand Murugappan and Ling Liu. An energy efficient middleware architecture for processing spatial alarms on mobile clients. *MONET*, , 2010.

[6] Nusrat Sultana, Tanzima Hashem, and Lars Kulik. Group nearest neighbor queries in the presence of obstacles. In *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, USA, 2014*, ACM, 2014.

[7] Jun Zhang, Dimitris Papadias, Kyriakos Mouratidis, and Manli Zhu. Spatial queries in the presence of obstacles. In *Advances in Database Technology - EDBT 2004, 9th International Conference on Extending Database Technology, Greece, 2004, Proceedings*, Springer, 2004.

APPENDIX

A. HEADINGS IN APPENDICES

The rules about hierarchical headings discussed above for the body of the article are different in the appendices. In the **appendix** environment, the command **section** is used to indicate the start of each Appendix, with alphabetic order designation (i.e. the first is A, the second B, etc.) and a title (if you include one). So, if you need hierarchical structure *within* an Appendix, start with **subsection** as the highest level. Here is an outline of the body of this document in Appendix-appropriate form:

A.1 Introduction

A.2 The Body of the Paper

A.2.1 Type Changes and Special Characters

A.2.2 Math Equations

Inline (In-text) Equations.

Display Equations.

A.2.3 Citations

A.2.4 Tables

A.2.5 Figures

A.2.6 Theorem-like Constructs

A Caveat for the TEX Expert

A.3 Conclusions

A.4 Acknowledgments

A.5 Additional Authors

This section is inserted by LATEX; you do not insert it. You just add the names and information in the `\additionalauthors` command at the start of the document.

A.6 References

Generated by bibtex from your .bib file. Run latex, then bibtex, then latex twice (to resolve references) to create the [5] .bbl file. [4] Insert that .bbl file into the .tex source file and comment out the command `\thebibliography`.

B. MORE HELPS

The sig-alternate.cls file itself is chock-full of succinct and helpful comments. If you consider yourself a moderately experienced to expert user of LATEX, you may find reading it useful but please remember not to change it.