Spatial Alarms In Obstructed Space*

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

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

ABSTRACT

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 high availability of smart phones has led to the proliferation of location based services. 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. 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.

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 in 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.

Existing work in this area has focused mainly on Euclidean distance and road network models, to the best of our knowledge; no work is yet done on spatial alarms in obstructed space. Spatial alarm evaluation in 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 vehicle can only go to a specific location following a predefined road, but a pedestrian's path is not limited by roads. However, a pedestrian is obstructed by various obstacles such as buildings or trees. So while calculating the distance from spatial alarms, we have to consider the obstructed distance.

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.

2. PROBLEM SETUP

Obstructed Space Route Problem denotes the problem of finding the shortest route between two query-points in Obstructed Space where non-intersecting 2D polygons repre-

*(Produces the permission block, and copyright information). For use with SIG-ALTERNATE.CLS. Supported by ACM.

WOODSTOCK '97 El Paso, Texas USA

sent *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 $D_{obs,a,b}$.
A **Spatial Alarm Query in Obstructed Space** is formally defined as follows:

DEFINITION 2.1. *Given the user's current location p,and an alarming distance U_d for an alarm, spatial alarm query returns the set of alarms A, where for each a$\in$ A,$D_{obs,p,a} < U_d$.*

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

DEFINITION 2.2. ***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}.

DEFINITION 2.3. ***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.*

DEFINITION 2.4. ***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.*

3. RELATED WORK

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 on the other hand in road networks navigation is limited along predefined roads.
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 [**?**],[**?**],[**?**] such as nearest neighbor and group nearest neighbor in obstructed space.
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.

4. NAIVE APPROACH

To compare the efficiency of our approach, we are about to represent two straightforward solutions for processing spatial-alarms in an obstructed space - sequential alarm (POI) processing in regular basis and region-divided alarm (POI) processing. Both of these algorithms are stated and explained in the following subsections.

4.1 Sequential Processing on Regular Basis

The POIs are saved in an R-tree indexed by own distance from the query point. In this naive approach, a maximum number of nearest POIs are retrieved from the server and on a minimum change of the user's location, a system event will fire an update procedure 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.

Algorithm 1 Fetch All POI

1: **procedure** FETCHPOI
2: From server, query to get P = {nearest 100 POI (at max) within a max. Euclidean distance A_{max} } and O = $\{p \in P$: all surrounding obstacles around p$\}$.
3: Construct the visibility graph V_G using the sets P and O
4: Let $k_{min} = A_{max} - U_{min}$, where U_{min} = minimum alarming distance among $\forall p \in P$.
5: Register the procedure UpdateNaive to be triggered on a minimal location-change of the client.
6: **end procedure**
7:
8: **procedure** UPDATENAIVE
9: If the client gets inside any POI's alarming distance, then alarm the client about the POI to be reached.
10: If k_{min} is crossed, then run the FetchPOI procedure again.
11: **end procedure**

The run-time of the first line of the second procedure UpdateNaive is $O(|P|)$, whereas the second line depends on the amount of location-change and the FetchPOI procedure.
Visibility graph construction requires $O(n^2)$ [**?**], where n = the number of edges of the obstacles. $O(|P|)$ run-time is needed to calculate U_{min} and above all, a huge overhead is sufficed to make P and O sets using such procedure.
Moreover, the accuracy of this naive approach is vulnerable to many cases which are explained and handled in the following approaches one by one.

4.2 Region-based Alarm Processing

This naive approach is a region-based modified straight forward approach which searches for a new alarm in the known region as soon as the client changes it's position. This naive algorithm works in three procedures - the first one retrieves all POI and obstacles and then computes the known and reliable regions, the second one calculates the visibility graph and periodically checks the client status to give alarms for any POI after every hibernation-period expiration and the third procedure checks the region crossings to recompute the answer set on any minimal location-change of the client.
In this approach, more than one query for the known region computation has to be done frequently to the sever after computing the visibility graph and if any unreachable

Algorithm 2 Region Based Alarm

procedure InitRegionBase
Make the location of the user as the center of the known region.
The server incrementally increases r_{known} until it finds at least 1 POI within this circle and makes P = set of all POI inside the circular region of radius r_{known}
Make O = Set of all obstacles inside the radius r_{known}
Construct the visibility graph V_G
Return the client a response with r_{known}, V_G, P, O
end procedure

procedure SafeRegionCalc
Construct the visibility graph for the set of obstacles and POIs supplied by the server in procedure InitRegionBase.
If any POI is excluded in the V_G, then query to the server again for expansion of the known region
Compute the obstructed distances $U_{min} = min$(obstructed alarming distance respective to all Euclidean alarming distance U_i) and similarly U_{max} as the maximum of those.
Compute the reliable region for the known region with radius, $r_{rel} = r_{known} U_{min}$.
If distance from any POI within the reliable region is less than U_i , trigger an alarm to the user, where U_i = the alarming zone's radius of the ith POI
Compute the safe region for the reliable region with radius, $r_{safe} = (D_{min} - U_{max})$
end procedure

procedure OnLocationChange
If the change in distance is less than r_{safe}, then return.
If the client is outside the reliable region, then re-run A and then B procedure and return.
If the safe region is crossed, then re-compute the reliable region including the visibility graph and run procedure InitRegionBase again.
end procedure

POI is found out - 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 so gets the approach less efficient. These problems are solved in the following final approach.

5. SPATIAL ALARMS IN OBSTRUCTED SPACE

Our spatial alarm evaluation system is divided into client-server architecture. The server has access to the locations of mobile users, location of alarms and location of obstacles. In this paper we assume that all users have access to some sort of localization service such as GPS or Wi-Fi that allow the server to pinpoint their current location. The client application is a thin-weight application that communicates with the server at regular intervals to retrieve necessary information about alarms and the obstacles. With the help of the client application the users can personalize their alarms as public, private or shared. They can join any public alarm or shared alarm as per their choice. We assume that the user can use any device such as smart-phones or PDA.
Our aim is to provide an energy-efficient, concise and accurate spatial alarm service for obstructed space. To preserve energy, the popular approach is to put the device in use to a low-power consumption state or sleep state. We propose a novel approach of computing a *safe region*. As long as the client device is in the safe region, the device can be put into sleep state without the risk of any alarm being missed.

Spatial alarm evaluation can be optimized using two key features: firstly, reducing the number of device wake-ups and second reducing the re-computation of same obstructed distance and reducing the number of duplicate data retrieval from the server. For the first strategy to be successful our safe-region computation should be accurate and optimal. We propose an algorithm in the section which will compute an optimal safe-region for our spatial alarm evaluation system. The second optimization technique is related to the safe-region computation technique. To compute the safe region the client application must communicate with the server as it needs the location of obstacles and alarms. In this paper we aim to optimize this communication by ensuring that no redundant data is retrieved from the server. We propose two different type of strategies which highlight exactly one of the aforementioned key features. Our application has two different modes,namely, *Bandwidth Saving Mode* and *Computational Cost Saving Mode*

5.1 Bandwidth Saving Mode

In this mode the main focus is to reduce the bandwidth of communication between the server and the client.In this mode the procedure is divided into three steps. *Initialization*,*Computing the Reliable Region and the Safe Region (CRRSR)* and *Update on location change*. Algorithm 3 shows the pseudo-code for initialization. The input to the algorithm is the location of the client, the outputs of the algorithms are radius of the known regions ($r_k p$,$r_k o$),the set of all obstacles within $r_k o$ and the set of all POIs within $r_k p$.

In the Initialization algorithm, the fact that the incremental-search will find at least 1 POI and stop the increment to get a fixed $r_k p$ 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

Algorithm 3 Initialization

1: **procedure** INITIALIZATION
2: Incrementally increase r_kp until there is at least 1 POI inside this radius
3: Make P = set of all POI inside the circular region of radius r_kp
4: Retrieve all the obstacles inside the circular region of radius r_kp
5: Let $r_ko = r_kp$
6: Construct the visibility graph V_G for the POIs
7: **while** there is any unreachable POI in the V_G and there are more than 1 collision of obstacle perimeter of the circle of radius r_ko **do**
8: expand rko until the unreachable POI gets reachable or there is less than 2 collisions
9: retrieve new obstacles;
10: If any new POI is found, then let $r_kp = r_ko$ and go to the 3rd step
11: **end while**
12: Make O = set of all obstacles inside the radius r_ko
13: Return the client a response with r_kp, r_ko, V_G, P, O
14: **end procedure**

completeness of this algorithm 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. □

In this procedure, the visibility graph is computed using the answer sets P and O got as return of the Init procedure from the server side. The most critical part of this algorithm is to calculate the safe region. This can be explained with a suitable example depicted in the figure-3. Figure **??** Safe region Computation

To prove the accuracy and completeness of the above algorithms, it is found to be enough to prove the following 3 facts.

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

Algorithm 4 Computing the Reliable Region and Safe Region

1: **procedure** CRRSR
2: Construct the visibility graph V_G for the set of obstacles and POIs supplied by the server while running the Init procedure
3: Compute reliable region for the known region: $r_{rel} = r_{ko} - U_{max}$ (fact: $r_{ko} \geqslant r_{kp}$)
4: Let D_i = obstructed distance of the ith POI from the user position according to V_G and U_i = alarming zone's radius of the ith POI according to V_G = alarming zone's radius of the ith POI according to V_G
5: Compute the radius of the safe region as $r_{safe} = min(D_i - U_i)$, for i = 1 to total no. of POIs
6: **while** a **do**ny POI falls inside r_{safe} and the full path of it in V_G also falls inside r_{safe} ,
7: revise $r_{safe} = min($ previous r_{safe} , Euclidean distance of that POI, $U_i)$
8: **end while**
9: If the user is inside the alarming zone of any POI, then trigger an alarm to the user for that POI getting reached
10: **end procedure**

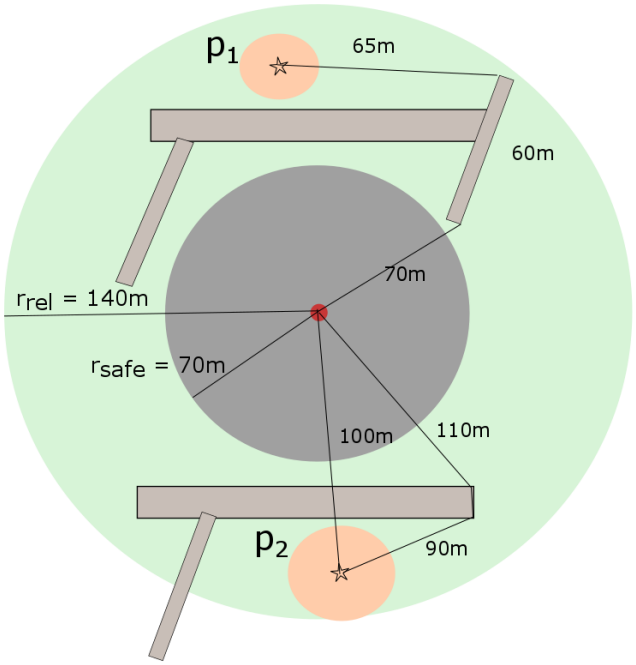

Figure 1: Safe region computation

Algorithm 5 Update On Location Changed

procedure UPDATE
if the client is outside the reliable region **then**
run Init and CRRCR algorithms and return
end if
if the safe region is crossed **then**
re-compute reliable region including V_G and run CRRSR
end if
if the user in alarming zone of a POI **then**
trigger an alarm to the user for that POI
end if
Let d = distance of the user from the center of the reliable region
Set the next minimum distance to trigger this procedure as min(r_{safe}, d)
end procedure

that mean, $r > |D - U|$ which is a contradiction because the procedure CRRSR 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, Procedure B computes the minimum among the Euclidean distances of the POIs. As we know from the Euclidean lower bound property that the obstructed distance>= Euclidean distance. 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, Procedure B choses 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 Init and CRRSR 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 Ui . But meanwhile, the user must cross the reliable region because $r_{ko} - r_{rel} = U_{max} > U_i$ So according to procedure C, procedure A and B 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. □

5.2 Computational Cost Saving Mode

The algorithm runs in this mode almost similarly as the "Bandwidth Saving Mode" with all the 3 described procedures - other than the Init procedure returning the computed V_G from the server side and the CRRSR procedure not reconstructing this V_G in the client side.

6. 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.

7. 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.

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.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 [**?**] .bbl file. [**?**] Insert that .bbl file into the .tex source file and comment out the command `\thebibliography`.

B. MORE HELP FOR THE HARDY

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.