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.

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 q,and an alarming distance u_d for an alarm, spatial alarm query returns the set of alarms P, where for each $p \in P, D_{obs,q,p} < u_i$.*

We define three different type of regions: *Known Region*, *Reliable Region* and *Safe Region* to provide spatial-alarm in an obstructed space to the client accurately and efficiently.

DEFINITION 2.2. ***Known Region:*** *We define two known regions for the POIs and the obstacles separately. 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 q and the current location as q', if $|q - q'| < r_{rel}$, 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. This region can also be called a hibernation-region, since the client just hibernates without doing any literal calculation while it's inside this region. We will denote the radius of the safe region as r_{safe}. Given the user's previous location q and the current location q', if $|q - q'| < r_{safe}$, then no calculation is needed to compute the answer set.*

The GETALLPOI(q, R_{max}, P_{max}) function populates the set P of POIs within maximum of R_{max} radius centring the query point (client's current location) q and with a maximum count of P_{max}. Similarly, GETOBSTACLESET(q, R_{max}, O) function returns the set of all obstacles within maximum of R_{max} radius centring q, but except those in a previous obstacle set O given as the third parameter.
Instead of the very naive GETALLPOI(q, R_{max}, P_{max}), a modified function CHECKNEWPOI(q, r_k, P) is used in our later approaches to retrieve all excluded POIs of the given set P within the radius r_k centring the point q.
MAKEVISGRAPH(P, O) function returns the *visibility graph* V_G with the set of POIs P and 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.
ALARMUSER(p_i) triggers an alarm to the user since the alarm (POI) p_i is reached and also marks p_i to be reached in the set of POIs P.
Moreover, EUCLIDEANDIST(q, q') returns the Euclidean distance between the two points q and q', whereas OBSDIST(q, q', V_G) returns the obstructed distance between the points q and q' computed from the visibility graph V_G.
In the final approach, two more functions returning boolean value are used as ISANYPOIUNREACHABLE(V_G) and ISPATHINSIDE(q, p_i, r_{safe}, V_G). 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. Another function, ISPATHINSIDE(q, p_i, r_{safe}, V_G) returns true if the POI p_i is inside the radius r_{safe} centring q within the visibility graph V_G along with all of its connecting edges from the center q, otherwise returns false.
Notations used in describing the algorithms follow the fact that all Euclidean distance related variables are in small letters whereas the variables related to obstructed distance are in capital letters.

3. RELATED WORK

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.
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.
Again, comprehensive research [2] 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 APPROACHES

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.

In this approach, the visibility-graph construction requires $O(n^2)$ [5], where $n =$ the number of edges of the obstacles and this algorithm is run more often from the update algorithm. A huge overhead is also sufficed to make P and O sets using such procedure. Again, The algorithm 5 can also be run much less time than in this approach, which is improvised in the later approaches.

Algorithm 1: InitNaive1(q, r_{max}, p_{max})

Input : Query point q, Max search radius r_{max}, max no. of POIs p_{max}
Output: Answer set A

$P \leftarrow$ GetAllPOI(q, r_{max}, p_{max})
$O \leftarrow$ GetObstacleSet$(q, r_{max}, \emptyset)$
$V_G \leftarrow$ MakeVisGraph(P, O)
$u_{min} \leftarrow \infty$
for $i \leftarrow 1$ *to* $|P|$ **do**
  $u_{min} \leftarrow min(u_{min}, u_i)$
$k_{min} \leftarrow (r_{max} - u_{min})$
return $A \leftarrow$ MakeAnswerSet(q, k_{min}, P, O, V_G)

Algorithm 2: UpdateNaive1(q', A)

Input : q', A

$q_d \leftarrow$ EuclideanDist(q, q')
for $i \leftarrow 1$ *to* $|P|$ **do**
  if EuclideanDist$(q', p_i) > u_i$ **then**
    AlarmUser(p_i)
if $q_d > k_{min}$ **then**
  call InitNaive1$(q', 100, 100)$

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 inside the known region as soon as the client changes it's position. This naive algorithm works in three parts - 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. The algorithms 3,4,5 show the respective parts.
The input to the algorithm 3 is the current location of the user q and the increment delta, the amount by which the regions will be expanded , r_d. The output of the algorithm is an answer set A which consists of the set of all obstacles O and POIs P along with the respective visibility graph V_G within the known-region radius r_k centring q.

Algorithm 3: InitRegionBase(q, r_d)

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

while $|P| < 1$ **do**
  $r_k \leftarrow (r_k + r_d)$
  $P \leftarrow$ CheckNewPOI(q, r_k, P)
$O \leftarrow$ GetObstacleSet$(q, r_k, \emptyset)$
$V_G \leftarrow$ MakeVisGraph(P, O)
return $A =$ MakeAnswerSet(r_k, P, O, V_G)

The input to the algorithm 4 is the current location of the client q and the answer set from the algorithm 3. This algorithm is also responsible for triggering an alarm to the client, if s/he is within the alarming distance of any POI.

Algorithm 4: SafeRegionCalc(q, A)

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

if isAnyPoiUnreachable(V_G) **then**
  $A \leftarrow$ InitRegionBase$(q, 10)$
  call SafeRegionCalc(q, A)
$u_{min} \leftarrow \infty, u_{max} \leftarrow 0$
for $i \leftarrow 1 to |P|$ **do**
  $D_i \leftarrow$ ObsDist(q, p_i, V_G)
  $D_{min} \leftarrow min(D_{min}, D_i)$
  $D_{max} \leftarrow max(D_{max}, D_i)$
$r_{rel} \leftarrow (r_k - D_{min})$
foreach $p_i \in P$ **do**
  if ObsDist$(q, p_i, V_G) < u_i$ **then**
    AlarmUser(p_i)
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), the radius of the safe region(r_{safe}), the reliable region(r_{rel}) and the known region(r_k) 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 5: OnLocationChange$(q, q', r_{safe}, r_{rel}, r_k)$

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

$q_d \leftarrow$ EuclideanDist(q, q')
if $q_d > r_{rel}$ **then**
  $A \leftarrow$ InitRegionBase(q, q_d)
  call SafeRegionCalc(q, A)
else if $q_d > r_{safe}$ **then**
  call SafeRegionCalc(q, A)
return $d_u = r_{safe}$

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 if any unreachable 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 ALARM 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 secondly 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. This mode is designed to operate in three parts - *client-initialization*, *alarm-configuration* and *update on any location change*. Algorithm 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. The output of the algorithm is an answer set A which consists of the radius of the known regions of POI 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 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 u_i is the alarming radius, d_i is the Euclidean distance from the center of the regions and D_i is the obstructed distance in the visibility graph V_G all for the i^{th} POI.

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

Algorithm 6: ClientInit(q, r_d)

```
Input  : Query point q, increment delta r_d
Output: The answer set, A = {r_kp, r_ko, P, O}
1  while |P| < 1 do
2      r_kp ← (r_kp + r_d)
3      P ← CheckNewPOI(q, r_kp, P)
4  r_ko ← r_kp
5  O ← GetObstacleSet(q, r_ko, ∅)
6  V_G ← MakeVisGraph(P, O)
7  while isAnyPoiUnreachable(V_G) do
8      r_ko ← (r_ko + r_d)
9      O ← GetObstacleSet(q, r_kp, O)
10     P′ ← CheckNewPOI(q, r_ko, P)
11     if |P′| > |P| then
12         P ← P′
13         r_kp ← r_ko
14         goto step-4
15 return A = MakeAnswerSet(r_kp, r_ko, P, O)
```

Algorithm 7: ConfigAlarm(q, A)

```
Input  : Query point q, answer set A
1  V_G ← MakeVisGraph(P, O)
2  r_rel ← (r_ko − u_max)
3  r_safe ← ∞
4  for i ← 1 to |P| do
5      r_safe ← min(r_safe, D_i − u_i)
6  foreach p_i ∈ P do
7      if D_i ⩽ u_i then
8          AlarmUser(p_i)
9      else if d_i < r_safe and isPathInside(q, p_i, r_safe, V_G)
       then
10         r_safe ← min(r_safe, d_i)
```

calculate the safe region. This can be explained with a suitable example depicted in the figure 3 and described in the subsection 5.3.

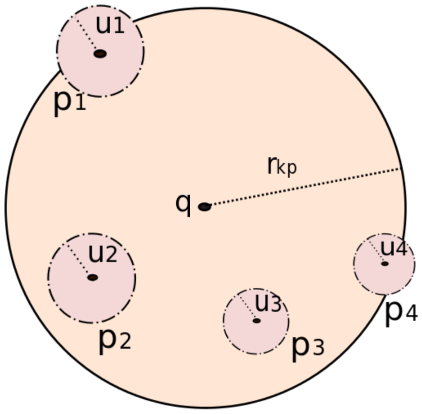

Figure 1: Retrieved POIs within r_{kp} centring q

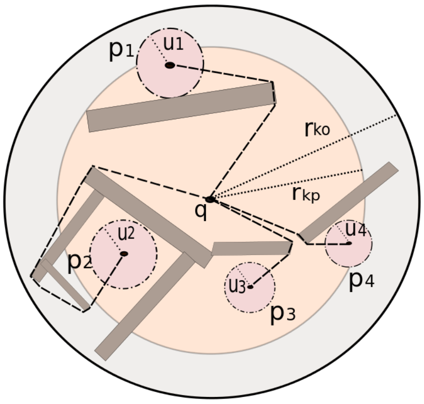

Figure 2: Visibility graph with all POIs and obstacles

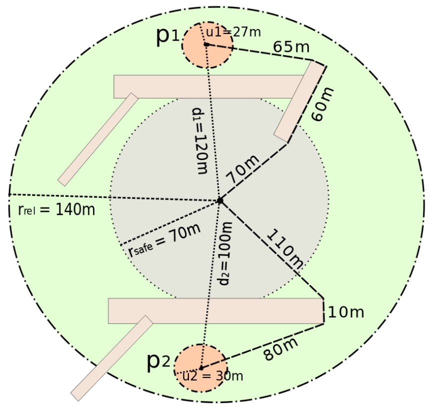

Figure 3: A really Awesome Image

The input to the algorithm 8 is the current and the previous location of the client (q' and q), the radius of the safe region(r_{safe}), the reliable region(r_{rel}) and the known region(r_k) 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($q, q', r_{safe}, r_{rel}, r_{kp}, r_{ko}, A$)

Input : $q, q', r_{safe}, r_{rel}, r_{kp}, r_{ko}$
Output: d_u

$q_d \leftarrow$ EuclideanDist(q, q')
if $q_d > r_{rel}$ **then**
  $A \leftarrow$ ClientInit(q, q_d)
  call ConfigAlarm(q, A)
if $q_d > r_{safe}$ **then**
  call ConfigAlarm(q, A)
return $d_u = r_{safe}$

5.2 Computational Cost Saving Mode

The algorithm runs in this mode almost similarly as the "Bandwidth Saving Mode" with all the 3 described parts - other than 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.

5.3 Example Simulation

In figure 1 an example scenario is put as if 3 POIs p_1, p_2, and p_3 are retrieved and in the figure 2 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 3, 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 2.4 , 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, $r_{safe} = min(93, 100-30) = 70m$.

5.4 Proof of Accuracy

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 re-

gion) 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 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, 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 Ui . 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. □

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.

8. REFERENCES

[1] Bhuvan Bamba, Ling Liu, 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), 22-26 June 2009, Montreal, Québec, Canada*, pages 207–214. IEEE Computer Society, 2009.

[2] Myungcheol Doo, Ling Liu, Nitya Narasimhan, and Venu Vasudevan. Efficient indexing structure for scalable processing of spatial alarms. In Divyakant Agrawal, Pusheng Zhang, Amr El Abbadi, and Mohamed F. Mokbel, editors, *18th ACM SIGSPATIAL International Symposium on Advances in Geographic Information Systems, ACM-GIS 2010, November 3-5, 2010, San Jose, CA, USA, Proceedings*, pages 426–429. ACM, 2010.

[3] Yunjun Gao, Jiacheng Yang, Gang Chen, Baihua Zheng, and Chun Chen. On efficient obstructed reverse nearest neighbor query processing. In Isabel F. Cruz, Divyakant Agrawal, Christian S. Jensen, Eyal Ofek, and Egemen Tanin, editors, *19th ACM SIGSPATIAL International Symposium on Advances in Geographic Information Systems, ACM-GIS 2011, November 1-4, 2011, Chicago, IL, USA, Proceedings*, 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 Christian S. Jensen, Christopher M. Jermaine, and Xiaofang Zhou, editors, *29th IEEE International Conference on Data Engineering, ICDE 2013, Brisbane, Australia, April 8-12, 2013*, pages 1372–1375. IEEE Computer Society, 2013.

[5] Anand Murugappan and Ling Liu. An energy efficient middleware architecture for processing spatial alarms on mobile clients. *MONET*, 15(4):543–561, 2010.

[6] Nusrat Sultana, Tanzima Hashem, and Lars Kulik. Group nearest neighbor queries in the presence of obstacles. In Yan Huang, Markus Schneider, Michael Gertz, John Krumm, and Jagan Sankaranarayanan,

editors, *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, Dallas/Fort Worth, TX, USA, November 4-7, 2014*, pages 481–484. ACM, 2014.

[7] Jun Zhang, Dimitris Papadias, Kyriakos Mouratidis, and Manli Zhu. Spatial queries in the presence of obstacles. In Elisa Bertino, Stavros Christodoulakis, Dimitris Plexousakis, Vassilis Christophides, Manolis Koubarakis, Klemens Böhm, and Elena Ferrari, editors, *Advances in Database Technology - EDBT 2004, 9th International Conference on Extending Database Technology, Heraklion, Crete, Greece, March 14-18, 2004, Proceedings*, volume 2992 of *Lecture Notes in Computer Science*, pages 366–384. 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 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.