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.

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

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.

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.

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

Known Region: We define two different known regions for the POIs and the obstacles. The region containing at least 1 POI is the known region for POI.
The region circulating the POIs known region containing none or single colliding obstacles is the known region for the obstacles. The set of obstacles and POIs within this region is known to the client. Both of the known regions are represented by a parabola whose focus point is the users location q,with the equation $y^2 = 4ax$ where $a = mr$ which are bounded by a straight line.

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

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

Table 1: Symbol Table

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

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

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

To compare the efficiency of our approach, we present a straightforward and naive solution for processing spatial-alarms in an obstructed space besides of our approach.

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

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

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

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

Algorithm 1: GetAlarmables(q, r)

```
  Input  : Query point $q$ and the search radius $r$
  Output: The visibility graph $V_G$
1 $P \leftarrow$ GetAllPOI$(q, r)$
2 $O \leftarrow$ GetObstacleSet$(q, r)$
3 $V_G \leftarrow$ MakeVisGraph$(P, O)$
4 return MakeAnswerSet$(q, V_G)$
```

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

Algorithm 2: UpdateClient(q, A)

```
  Input  : Client's current location $q$, latest answer set $A$
1 foreach $p_i \in P$ do
2     if $dist_O(q, p_i, V_G) > p_i.u$ then
3         AlarmUser$(p_i)$
4 if $dist_E(A.q, q) > A.d_u$ then
5     GetAlarmables$(q, 100)$
```

In this naive-approach, the visibility-graph is constructed more times than necessary to hold accuracy, each of which constructions requires $O(n^2)$ [5], where $n =$ the number of edges of the obstacles. A huge overhead is also sufficed to make P and O sets using such procedure. Again, the Algorithm 4 can also be run much less time than in this approach, which is improvised in the later approach.

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 Computation of the regions

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

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

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

6.2 Bandwidth Saving Mode

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

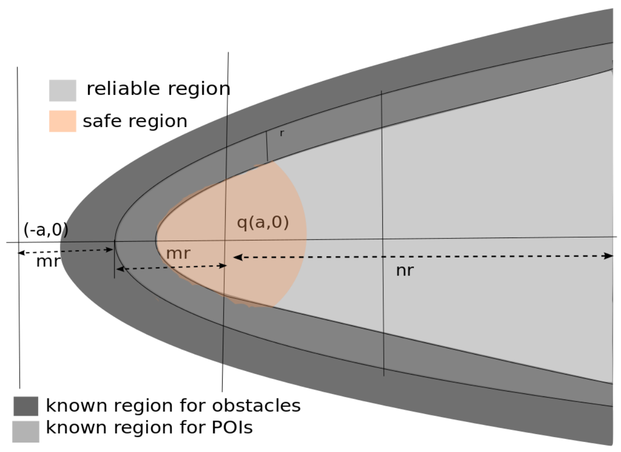

Figure 1: Different regions

The input to the client-initialization algorithm (Algorithm 3) is the current location of the client q, query radius r, current velocity of the client v, path history as a set of straight lines $S = (m_1, c_1, l_1), (m_2, c_2, l_2), (m_3, c_3, l_3), \dots\ (m_n, c_n, l_n)$ and the last answer set $A_{prev} = q,\ u,\ m_p,\ m_o,\ n$, where u is the vertex of the . This algorithm makes the frequent queries more efficient and accurate and needs much less server communication.

In the following algorithms, PREDICTDIRECTION(S) function returns a vector by inerpolating all the line segments given in a set S as the client's path history. During the linear interpolation, the oldest path is given the minimum weight, while the later ones get the incrementally higher weights iventually giving the latest path segment the maximum weight and predicting the client's current direction most likely to be towards the latest paths.

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

The function VARIANCEOFPATH(S) returns a real number within the range $[2, \infty)$ which is proportional to the variation of the client's path directions given the directed path segments in the set S. This value is assigned to the multiplier m_p to construct the parabola $y^2 = 4 * (m_p * r) * x$, since the more variation in the client's path requires more cross-section area of the known region parabola.

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

GETCOLLISIONPOINT(O, q, m_o, r, n_o) checks any collision of any of the obstales from the set O with the parabola $y^2 = 4 * (m_o * r) * x$ and also the bounding straight line at distance $(n_o * r)$ from the focus q and perpendicular to the axis of the parabola. Then it returns an all negative point if there is no collision, otherwise returns a positive point in the space. The isPositivePoint(p) reutrns true for the point p to be positive, otherwise returns false. Again, the function pointOnStraightLine (p, n, r, q) returns true if the point p is on the straight line at distance $(n * r)$ from the focus q and perpendicular to the axis of the parabola.

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

The output of the Algorithm 3 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 3: QueryAlarmables(q, r, v, S, A_{prev})

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

The input to the Algorithm 4 is the current location of the client q and the answer set got from the Algorithm **??**. This algorithm is also responsible for triggering an alarm to the client if the client is within the alarming distance of any POI.

Algorithm 4: ConfigUpdate(q, A)

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

In this algorithm, the visibility graph is computed using the answer sets P and O got as return of the 3 from the server side.

The input to the algorithm 5 is the current location of the client q, the radius of the safe region(r_{safe}) and finally the already computed answer set A. The output of the algorithm is the minimum distance d_u to trigger this algorithm the next time.

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

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

6.3 Computational Cost Saving Mode

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

However, the main difference with the previously descried mode from this mode is - the Algorithm 3 returning the computed V_G as another element of the answer set A from the server side and the algorithm 4 not reconstructing this V_G in the client side during running the Algorithm 4, whereas the Algorithm 5 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 3.

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 `\addition-alauthors` 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.