Supporting Continuous Range Queries in Indoor Space

Wenjie Yuan & Markus Schneider
Department of Computer & Information Science & Engineering
University of Florida
Gainesville, FL 32611, USA
{wyuan, mschneid}@cise.ufl.edu

***Abstract*—Range query processing is an important technology in spatial databases. Current studies mainly focus on range queries in outdoor space based on the Euclidean or network distances, but rarely any approach is proposed for indoor space. In this paper, we present two network-based approaches to supporting stationary range queries and continuous range queries for indoor space. There are several challenges involved in designing these approaches. First, having no explicit network construction in indoor space makes it difficult to precisely determine the distances between objects. Second, users can walk to anywhere in the building, which makes it hard to predict users' movements for continuous range queries. Our proposed approaches can overcome these challenges and efficiently support range queries in indoor space.**

I. Introduction

Approaches for efficient range queries in outdoor space are of a particular interest in *Geographic Information Systems* (GIS). By using range queries, users are able to find the objects of their interests nearby. In recent years, the demands on processing range queries in indoor space have been witnessed. For example, during a holiday shopping season, it is very common for users to ask for stores of their interests within a certain distance. However, few approaches have been proposed to support range queries in indoor space.

There are several challenges involved in designing approaches to support range queries in indoor space. First, the process of range queries depends on the shortest reachable distances between objects. However, since indoor space is composed of cells instead of explicit networks, the precise shortest distance between two objects cannot be easily determined. Second, a network connecting all objects is required for answering range queries. However, it is hard to build such a network for indoor space. Although there are some comparable concepts in indoor space and outdoor space like corridors and roads, in indoor space, there are also concepts like rooms and lobbies for which we do not find counterparts in outdoor space. For example, inside a room, there are many implicit paths from one location to another. This makes it difficult to construct networks for indoor space. Third, in outdoor space, the movements of vehicles are restricted to roads. However, in indoor space, there is no constraint in users' movements, which causes difficulties in predicting users' next directions.

This work was partially supported by the National Science Foundation under grant number NSF-CAREER-IIS-0347574 and NSF-IIS-0812194.

In this paper, we propose two approaches to supporting stationary range queries and continuous range queries in indoor space. The two approaches are based on a *Direct Path Graph*(*DPG*), which is designed to construct a virtual path network for the entire indoor space and support a length-dependent optimal routing. If users' movements are restricted to these virtual segments, the existing approaches to range queries for outdoor space can be directly applied to the DPG to support range queries in indoor space. However, this assumption does not hold in most cases, and users can go any reachable place inside buildings. To solve this problem, our approach to processing stationary range queries applies a preprocess to connect the query point to the DPG, and the approach to continuous range queries divides cells into regions. The obtained interesting objects changes only when users walk from one region to another. Thus, by recording the boundary of each region, the changes of the qualifying objects can be managed.

The rest of the paper is organized as follows. Section II discusses the available approaches to range queries for outdoor space. Section III discusses the way we construct the network for indoor space. In Section IV and Section V, we introduce our approach to supporting range queries and continuous range queries in indoor space. Finally, Section VI draws some conclusions and depicts future work.

II. Related Work

In this section, we overview previous work related to stationary range queries and continuous range queries in outdoor space.

A. Stationary Range Queries

A stationary range query asks for interesting objects within a given distance with respect to a static query point. Approaches supporting stationary range queries can be subdivided into two categories: *Euclidean-based* approaches and *network-based* approaches. Euclidean-based approaches check the distances between objects by their relative positions in space while network-based approaches consider their actual reachable distances.

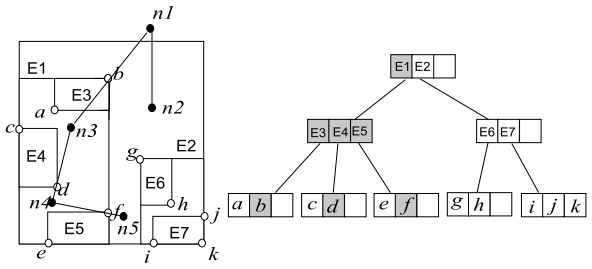


Fig. 1. An example of the Range Network Expansion approach

1) Euclidean-based Approaches: The most typical *Euclidean-based* approaches proposed in [3], [7], [5] use R-Trees [2], R^+-Trees [8], R^*-Trees [1] for query processing. Due to its simplicity and efficiency, the R-tree has become a popular index structure for Euclidean-based approaches.

2) Network-based Approaches: More practical solutions to support range queries are based on spatial networks. [6] proposes a network-based approach, called *range network expansion* (*RNE*), to support stationary range queries. It first expands the network from the query point, and selects all segments whose distance to the query point is less than a given range. Then for the selected segments, it picks the qualifying nodes on the R-tree (e.g. the gray nodes in Figure 1).

Using Voronoi diagrams is another choice of network-based approaches. [9] first introduces an approach to convert the network into a Voronoi diagram. This idea is then used by the VN^3 model proposed in [4], and the *PINE* model proposed in [10]. In [10], the authors mention that PINE requires less disk access time and less CPU time than VN^3.

B. Continuous Range Queries

Approaches for stationary range queries assume that the query point is always in a static position. In fact, users are usually in a moving status when they issue a navigation query. The importance of continuous queries is mentioned by Sistla et al. in [11]. In recent years, more and more approaches supporting continuous range queries have appeared. These approaches also can be classified into Euclidean-based approaches and network-based approaches.

1) Euclidean-based Approaches: An earlier Euclidean-based approach introduced in [12] first selects several sample points on the path, and then performs stationary range queries on each selected sample point.This approach suffers from a performance and accuracy problem: the more sample points we select, the more accurate but poorer the performance will be. The less sample points we take, the better but less accurate the performance will be. In [13] Tao et al. propose an idea of finding *split points* which indicate the changes of the qualified objects, to support continuous nearest neighbor queries. As shown in Figure 2, a and b are two interesting objects on the way from the start S to the end E. The split point, which is determined by the position of a and b, indicates that the nearest object exchanges from a to b. The approach proposed in [14] (shown in Figure 3) first determines all possibly qualified objects by drawing a region according to the given range

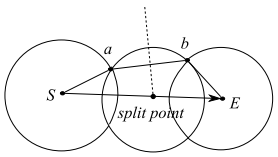


Fig. 2. An example of continuous query using split point

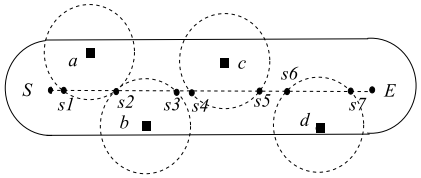


Fig. 3. An example of Euclidean-based continuous range query

(indicated by solid lines). Then, the split points $(s_1, s_2, \ldots, s_7)$ are determined by the intersections between the path and the dashed circles. The distances from the center point of each circle to its related split points are the given range.

2) Network-based Approaches: To the best of our knowledge, the approach proposed in [14] is the only one for supporting continuous range queries based on network distances. This approach is motivated by the idea of continuous K-nearest neighbor queries proposed in [15] and [16]. As shown in Figure 4, this approach first selects a segment that contains no intersection node (e.g. AB). Then it incrementally checks the nodes in nearby segments within a given range. The split points are determined by checking the condition $Sp[A, x] > dis[A, B]$, where x is an interesting object, $dis[A, B]$ is the distance between A and B, and $Sp[A, x]$ is computed by the formula *given range*$-dis[A, x]$.

In Euclidean-based approaches, the objects are evaluated based on their Euclidean distances and not on the actually reachable distances. Considering the walls separating the indoor space, it is impossible to apply Euclidean-based approaches to indoor space. The network-based approaches provide the ability of finding the exact network distances. Thus, they are superior to the Euclidean-based approaches when considering their practical utilization. However, none of them can be directly applied to indoor space for accurate computation of range queries. In this paper, we will propose two approaches to supporting continuous range queries in indoor space respectively.

III. The iNav Model: Constructing the Network

Range queries are used to find interesting objects within a certain distance. Thus, the reachable distances between any two objects are very important for range queries. However, indoor space is composed of different kinds of cells, such as rooms, corridors, and lobbies. Thus, cells are the basic units in indoor space. But the lack of detailed distance information between any two cells in a cell-based structure makes it difficult to directly apply the existing range query approaches to indoor space. In [18], we propose a conceptual indoor navigation model, called iNav, which can convert a cell-based

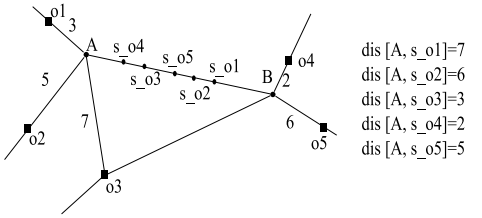


Fig. 4. An example of network-based continuous range query

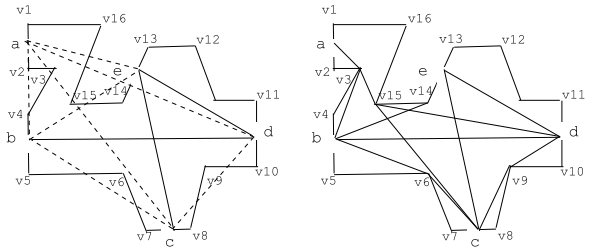


Fig. 5. Examples of path segment construction: lines that connect two boundary access points (a), final path segments in the polygon (b)

structure into a network structure, and support the shortest path search in indoor space.

Although there are no explicit path segments in indoor space, there are implicit path segments that people often take. For example, people like to go straight to a destination in case they can get there without any barrier. In the iNav model, these implicit path segments are determined according to the shapes of cells and the locations of their *access points*, which are the architectural constraints controlling the accessibility of cells (e.g. doors). According to cells' features and functions, the iNav model classifies them into four kinds of categories: *simple cell*, *complex cell*, *open cell* and *connecter*. Simple cells are the cells that are closed by walls and can be accessed by only one access point. Complex cells are the ones that are closed by walls and can be accessed by multiple access points. Open cells are the cells for which at least part of their boundaries is not given by explicit walls or other constraints. Connectors are objects that connect different floors in a building, such as stairs and elevators.

Since complex cells have multiple access points, they can serve as passages in indoor space. When they function as passages, the construction of path segments depends on the locations of their access points. Figure 5 is an example of a complex cell with five access points *a, b, c, d*, and *e*. For the access points that can be straightly connected without intersecting a wall, the implicit path segments are the straight lines connecting them (e.g. (c, e) and (d, e) in Figure 5b). For the access points that cannot be straightly connected, one or several vertices of the region are selected as the *intermediate points* to partition the original segment into several path segments (e.g. the vertex v_6 is the intermediate point that divides bc into bv_6 and v_6c). If an open cell serves as a target object, it is reasonable to consider the center point of its open boundaries as an access point. However, if it serves as a passage, the widths of its open boundaries may affect the final route. Thus, we first combine all connected open cells into one big complex cell, and find all the path segments in this cell. Then, find all the path segments in each individual open cell (as shown in Figure 6). Path segments in a connecter are segments connecting each pair of access points.

Algorithm 1 proposed here is the implementation of the conceptual model in [18]. It shows how to compute all path segments in a cell whose shape is given by a polygon. First, the lines connecting two access points are formed. Then we check whether the interior of any line intersects the boundary of this polygon or not. For example, Figure 5a shows the connections between any two access points on the boundary of the polygon. The interior of the segments (*a, b*), (*a, c*), (*a, d*), (*a, e*), (*b, c*), (*b, e*) and (*c, d*) all intersect the boundary. These segments must be divided into several path segments inside the polygon. From computational geometry [17] we know that if the interior of a line connecting two boundary points of a polygon intersects the boundary, this polygon must be a concave polygon. This means that there is at least one vertex whose interior angle is a reflex angle (degree $> 180°$). We call this kind of vertex *concave vertex*. The approach to find all path segments for which intersections happen is composed of two steps. One is to construct all potential path segments among all the access points and concave points (lines 4 to 8). Then the *Floyd-Warshall* algorithm is applied to compute the shortest path segments between any two access points (line 9).

Algorithm 1: Path Segments Construction

Input: polygon P, list<point> *accessPoints*, which is the list of access points on P's boundary.
Output: list<line> *pathSegments* which is the list of all path segments in P; list<points> *accessPoints*, which is the list of all the final access points.

```
1  list<line> pathSegments=new list<line>;
2  for each pair of points in accessPoints, check the
   intersection with the boundary of P;
3  if intersection happens then
4      list<line> potentialPS=new list<line>();
5      list<point> concaveVertices=find all the concave
       vertices in P;
6      for any two points a and b from concaveVertices or
       accessPoints do
7          if the line connecting a and b does not intersect
           the boundary of P then
8              add line(a,b) into potentialPS;
9      Floyd-Warshall(accessPoints, potentialPS);
10 else
11     pathSegments=the connections between all pairs of
       the access points;
```

The worst runtime complexity for identifying the intersections is $O(n^3)$, where n is the number of access points. Computational geometry [17] tells us that the complexity of finding all potential path segments in Algorithm 1 is

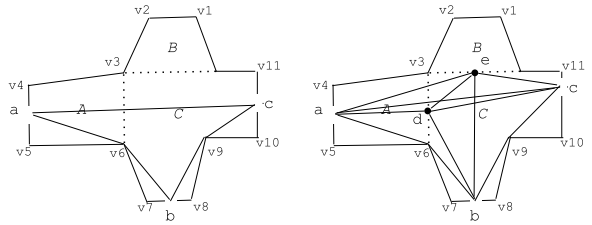


Fig. 6. Constructing path segments in open cells

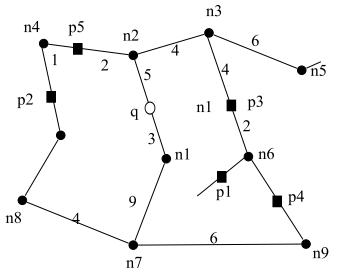


Fig. 7. An example of Incremental Network Expansion

$O((n+m)^2 \lg(n+m))$, where m is the number of concave vertices. Since the complexity of the Floyd-Warshall algorithm to compute the shortest paths between all pairs of nodes is $O((n+m)^3)$, the total complexity of Algorithm 1 is $O((n+m)^3)$.

The combination of the path segments in all cells forms a network, called *Direct Path Graph* (*DPG*), for the entire indoor space. The availability of each edge in the graph is controlled by the accessibilities of its two end points and the edge's interior. The DPG has several nice properties that are inherited from the path segments and can support range queries. First, a DPG represents the whole structure of implicit path segments with their accessibilities for the indoor space scenario. Second, because each path segment is a straight line between two points, the length of the path segment is calculated by using the Euclidean distance $\sqrt{(x_1-x_2)^2+(y_1-y_2)^2}$ where (x_1, y_1) and (x_2, y_2) are the coordinates of the two end points. Third, because all the path segments in each cell are the shortest ones, we can obtain the shortest distance between any two cells.

IV. Supporting Stationary Range Queries

In Section III, we have demonstrated how to construct the network structure for indoor space. The construction of the network has two important features that can nicely support range queries:

- All cells in indoor space are represented by at least one node in a DPG.
- For any node in the network, we can find the shortest distances between this node and all other nodes.

The first feature makes sure that no cell will be missing in the graph, and the second feature ensures that it is possible to find all the cells that are reachable within a given distance. In this section, we propose an *Indoor Range Network Expansion* (*IRNE*) algorithm to support stationary range queries in indoor space. This IRNE algorithm utilizes the idea of the *Incremental Network Expansion* proposed in [6], and extends it to fit the indoor environment.

A. Incremental Network Expansion

The *Incremental Network Expansion* (*INE*) approach presented in [6] is a network-based approach used for the *Nearest Neighbor Search*. Its basic idea is to expand the network segments from the query point to obtain the qualifying objects. As shown in Figure 7, assuming a user wants to find the nearest object of interest (denoted by black rectangles) from q, INE will first check the objects in the segment n_1n_2. Then, the segment n_1n_7 emanating from n_1 will be checked, followed by the segments n_2n_3 and n_2n_4 emanating from n_2. The expansion process will continue until the nearest qualifying object is found. Finally, p_5 is returned as the final result.

B. Indoor Range Network Expansion

Once we have built the DPG for the entire indoor space, we can process range queries by applying the INE algorithm to the graph. However, since users' movements are not restricted to the segments of the DPG, it is difficult to determine the starting segment. In order to overcome the problem, the *Indoor Range Network Expansion* (*IRNE*) algorithm (shown in Algorithm 2) takes two steps to compute the range queries. In the first step, if the point is not on any segment of the graph, we preprocess the query point by connecting it to any of the access points inside the cell with the shortest reachable path segments (lines 1 to 2). In the second step, segments are expanded by using a similar idea of INE, and all the interesting objects on the visited segments are selected as the final result (lines 3 to 12).

Algorithm 2: Indoor Range Network Expansion

Input: point $QueryP$; polygon P, list<Line> *pathSegments*, list<point> *accessPoints*.
Output: list<object> *interesting_objects*.

```
1  find all the shortest path segments from QueryP to all
   the other access points in this cell by using
   Path_Segments_Construction algorithm;
2  insert all the adjacent nodes of QueryP into Q
   according to their distances to QueryP;
3  de-queue the node n in Q with the smallest
   Dis(QueryP, n);
4  while Q is not empty do
5      foreach non-visited adjacent node n_x of n do
6          if Dis(QueryP, n_x)<Range then
7              if n_x is an access point of A then
8                  //A is an interesting object;
9                  add A to interesting_objects;
10                 en-queue(n_x, Dis(QueryP, n_x));
11     de-queue the next node n in Q;
12 Sort interesting_objects to remove duplicates;
```

In order to explain this algorithm clearly, we use Figure 8 as an example of IRNE. Figure 8 is an indoor structure with

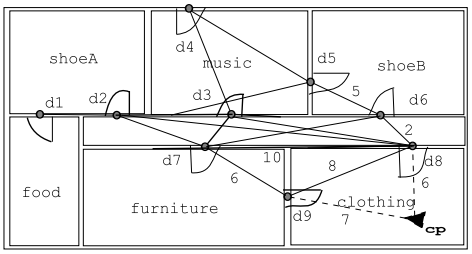
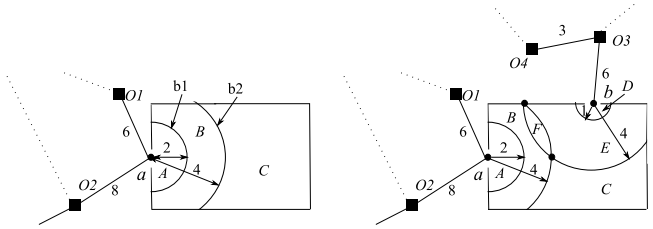


Fig. 8. An example of INE



(a) (b)

Fig. 9. A subdivision (a), and the overlay of two subdivisions (b)

its constructed DPG. The numbers near the lines represent the distances between the corresponding access points (e.g. the value 6 near $d7$ means that the distance between $d7$ and $d9$ is 6). We assume that all the access points are accessible at that time and the user's current position is represented by the triangle in the *clothing* store. For the query "find all the shoe stores within 10 meters", the IRNE algorithm first calculates the temporal path segments from the query point to all the access points in the *clothing* store, which are represented by the dashed lines in Figure 8. Because d_8 and d_9 are adjacent nodes of the query point, and their distances to the query point are less than the given range, d_8 and d_9 are inserted into Q. We first pop d_8 from Q, and check all its adjacent nodes to see if some of them are the access points of a shoe store. Since d_6 is an access point of *shoeB*, *shoeB* is selected as a qualifying object. Then we insert the nodes that are adjacent to d_8 and whose distances to the query point are less than the given range (e.g. d_6). We continue to pop nodes from Q and perform the same process to each of the popped nodes until Q is empty. Finally, *shoeB* is selected as the result.

The time complexity of constructing path segments from the current position is $O((m+n)^3)$, where n is the number of access points in the current cell, and m is the number of concave vertices in the polygon. In fact, the value of m and n in the real world are relatively small in most cases. Thus, the complexity of Algorithm 2 relies on the number of path segments it expands, and the worst case is $O(|E|)$.

V. Supporting Continuous Range Queries

Traditionally, approaches supporting continuous range queries try to find intervals in which the qualifying objects remain the same. When the query point moves from one interval to another, the qualifying objects change accordingly. However, these approaches are not appropriate for the cell-based structure in indoor space. In this section, we propose the *Indoor Range Region Division* (*IRRD*) approach to support the continuous range queries in indoor space.

A. The Indoor Range Region Division Algorithm

The idea of the IRRD approach is to divide each cell into several regions. In any position of a region, the set of qualifying objects that can be obtained are the same; and in two different regions, the qualifying objects are always different. The steps of the IRRD approach are shown in Algorithm 3:

Algorithm 3: Indoor Range Region Division

Input: list<line> *pathSegments*; polygon P; list<point> *accesspoints*
Output: list<region> R

```
1 list<object> qObject=new list<object>();
2 foreach access point ap in the current cell do
3     qObject=all interesting objects that can be reached
      from ap within the given range;
4     for i=qObject.size() to 0 do
5         create the region r_i according to the distance
          from ap to qObject.get(i);
6         assign qObject.get(i) to the result of all the
          existing regions;
7     create a subdivision containing all the regions of ap;
8 compute the intersection of all the subdivisions and
  assign the corresponding objects into each region;
```

We use Figure 9 to illustrate the IRRD algorithm. O_1, O_2 and O_3, O_4 in Figure 9(b) are the qualifying objects that can be reached from a and b. Then, for the access point a, we first choose the interesting object that has the longest distance from a (e.g. O_2) to construct the region in the cell. The region A with the boundary b_1 in Figure 9(a) is O_2's corresponding region, which is determined according to the distance between a and O_2. Then O_2 is added to the result set of A. The boundary of the second region B is determined according to O_1. Its boundary is composed of b_1 and b_2, where b_2 is determined according the radius of region A and the distance of O_1 to a. Thus, b_2 is constructed by expanding b_1 by $8-6=2$. O_1 is then added into the result sets of both A and B. Finally, the subdivision of access point a is constructed as shown in Figure 9a. The subdivision according to b can be obtained by using the same strategy. The final division is obtained by computing the overlay of the two subdivisions (as shown in Figure 9b). From Figure 9b, we can learn that in any position of a region, the obtained objects are always the same; and the result changes when users go from one region to another.

The construction of the subdivision for each access point (lines 2 to 7) is $O(m)$, where m is the number of qualifying objects reachable from the corresponding access point. From computational geometry [17] we know that the complexity of computing the overlay among the subdivi-

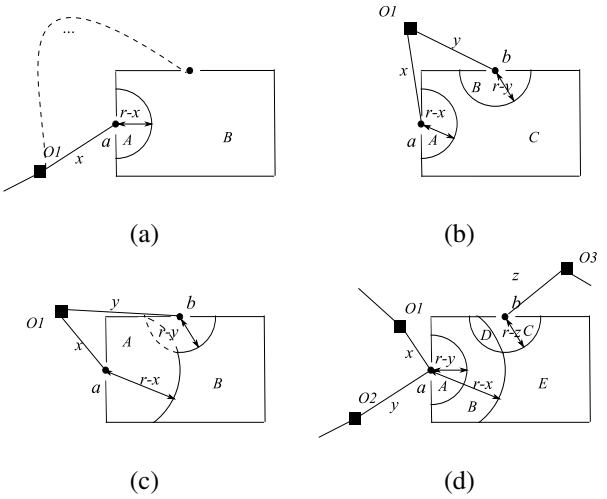


Fig. 10. The proof of Lemma 1

sions (line 8) is $O(n \lg n + k \lg n)$, where n is the total number of vertices in all the subdivisions, and k is the total number of intersecting points. Therefore, assuming there are p access points in a cell, the total complexity of Algorithm 3 is $O(pm + n \lg n + k \lg n)$.

Lemma 1 *The result of the qualifying objects will change and only change at the boundaries of each region calculated in the IRRD algorithm.*

Proof: Case 1: A single circle intersects the cell. In Figure 10a, x is the shortest distance from the access point a to object O_1. r is the given range distance. Region A is generated according to the location of a and the value $r - x$. From the figure we can see that from any position inside A, we can get to O_1 through a with the distance less than r; while from any position inside B, the distance to O_1 is always greater than r. Thus, the result changes on the boundary of A.

Case 2: Multiple circles corresponding to the same object do not overlap. If the distances from O_1 to a and b are less than r, there will be two corresponding circles in the cell. Figure 10b shows the scenario that the two circles do not overlap. When the user is inside A, O_1 is included in the final result. If he/she walks from A to C, O_1 will be removed from the result. When he/she walks into B, O_1 will be qualifying again. Thus, the result changes on the boundary of each region in the cell.

Case 3: Multiple circles corresponding to the same object overlap. In Figure 10c, the two circles corresponding to the same object O_1 overlap in the cell. The combination of the two circles forms a new region A. From any position of A, O_1 is a qualifying object, while from any position outside A, O_1 is not included in the result. Therefore, the result changes on the boundary of region A.

Case 4: Circles corresponding to different objects overlap. In the above cases, we have shown that for any case of a single object, Lemma 1 holds. This means that at least one object will be added or removed from the result when the query point moves from one side of a boundary to the other side. Since all the boundaries of the regions are composed of the boundaries of the circles, at least one object will be added or removed at the boundary of the regions (as shown in Figure 10d). ■

VI. Conclusions and Future Work

In this paper, we have proposed the IRNE and IRRD algorithms to compute stationary range queries and continuous range queries respectively for indoor space. These two algorithms utilize the DPG, which is designed to construct the network for the entire indoor space, to obtain the reachable distances between cells. By using the IRNE and IRRD algorithms, the range queries can be answered based on the network distances.

The range query type is only one of the important queries for navigation. In the future, we will explore how to support other important queries like the K-nearest neighbor query, and the closest-pairs query in indoor space.

References

[1] N. Beckmann, H. Peter Kriegel, R. Schneider, and B. Seeger. The R*-tree: An Efficient and Robust Access Method for Points and Rectangles. In *SIGMOD*, pages 322–331, 1990.

[2] A. Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. In *SIGMOD*, pages 47–57, 1984.

[3] G. R. Hjaltason and H. Samet. Distance Browsing in Spatial Databases. *ACM Trans. Database System*, 24(2):265–318, 1999.

[4] M. Kolahdouzan and C. Shahabi. Voronoi-Based K Nearest Neighbor Search for Spatial Network databases. In *VLDB*, pages 840–851, 2004.

[5] F. Korn, N. Sidiropoulos, C. Faloutsos, E. Siegel, and Z. Protopapas. Fast Nearest Neighbor Search in Medical Image Databases. In *VLDB*, pages 215–226, 1996.

[6] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao. Query Processing in Spatial Network Databases. In *VLDB*, pages 802–813, 2003.

[7] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest Neighbor Queries. In *SIGMOD*, pages 71–79, 1995.

[8] T. Sellis, N. Roussopoulos, and C. Faloutsos. The R+-Tree: A Dynamic Index For Multi-Dimensional Objects. In *VLDB*, pages 507–518, 1987.

[9] A. Okabe, B. Boots, K. Sugihara, and S. N. Chi. Spatial Tessellations: Concepts and Applications of Voronoi Diagrams. Wiley, UK, second edition, 2000.

[10] M. Safar. K Nearest Neighbor Search in Navigation Systems. *Mobile Information System*, 1(3):207–224, 2005.

[11] A. Prasad Sistla, O. Wolfson, S. Chamberlain, and S. Dao. Modeling and Querying Moving Objects. In *Int. Conf. on Data Engineering*, pages 422–432, 1997.

[12] Z. Song and N. Roussopoulos. K-Nearest Neighbor Search for Moving Query Point. In *7th Int. Symp. on Advances in Spatial and Temporal Databases*, pages 79–96, 2001. Springer-Verlag.

[13] Y. Tao, D. Papadias, and Q. Shen. Continuous Nearest Neighbor Search. In *VLDB*, pages 287–298, 2002.

[14] K. Xuan, G. Zhao, D. Taniar, and B. Srinivasan. Continuous Range Search Query Processing in Mobile Navigation. *Parallel and Distributed Systems*, 0:361–368, 2008.

[15] M. R. Kolahdouzan and C. Shahabi. Continuous K-Nearest Neighbor Queries in Spatial Network Databases. In *Int. Conf. on Scientific and Statistical Database Management*, pages 33–40, 2004.

[16] D. E. Maytham Safar. eDAR: Algorithm for Continuous KNN Queries Based on PINE. *Information Technology and Web Engineering*, 1, 2006.

[17] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. Computational Geometry: Algorithms and Applications. Springer, Berlin, 3rd ed. edition.

[18] Wenjie Yuan and Markus Schneider. iNav: An Indoor Navigation Model Supporting Length-Dependent Optimal Routing. In *the 13th AGILE Int. Conf. on Geographic Information Science*, 2010.