Probabilistic Threshold k Nearest Neighbor Queries over Moving Objects in Symbolic Indoor Space

Bin Yang[1,2] Hua Lu[1] Christian S. Jensen[1]

[1]Department of Computer Science, Aalborg University, Denmark
[2]School of Computer Science, Fudan University, China
byang@fudan.edu.cn, luhua@cs.aau.dk, csj@cs.aau.dk

ABSTRACT

The availability of indoor positioning renders it possible to deploy location-based services in indoor spaces. Many such services will benefit from the efficient support for k nearest neighbor (kNN) queries over large populations of indoor moving objects. However, existing kNN techniques fall short in indoor spaces because these differ from Euclidean and spatial network spaces and because of the limited capabilities of indoor positioning technologies.

To contend with indoor settings, we propose the new concept of minimal indoor walking distance (MIWD) along with algorithms and data structures for distance computing and storage; and we differentiate the states of indoor moving objects based on a positioning device deployment graph, utilize these states in effective object indexing structures, and capture the uncertainty of object locations. On these foundations, we study the probabilistic threshold kNN (PTkNN) query. Given a query location q and a probability threshold T, this query returns all subsets of k objects that have probability larger than T of containing the kNN query result of q. We propose a combination of three techniques for processing this query. The first uses the MIWD metric to prune objects that are too far away. The second uses fast probability estimates to prune unqualified objects and candidate result subsets. The third uses efficient probability evaluation for computing the final result on the remaining candidate subsets. An empirical study using both synthetic and real data shows that the techniques are efficient.

Categories and Subject Descriptors

H.2.2 [**Database Management**]: Physical Design—*access methods*; H.2.8 [**Database Management**]: Database Applications—*spatial databases and GIS*

General Terms

Algorithms, Experimentation, Performance

Keywords

Indoor Moving Objects, Probabilistic Threshold k Nearest Neighbor Queries, Symbolic Indoor Space, Uncertainty

1. INTRODUCTION

The availability of indoor positioning renders it possible to support interesting queries over large populations of moving objects in indoor spaces such as shopping malls, airports, subway stations, and office buildings. For example, support for k nearest neighbor (kNN) queries over indoor moving objects enables the detection of approaching potential threats at sensitive locations in a subway system; and it is possible for shops in an airport to target nearby individuals in promotions.

Two factors render existing kNN techniques in spatial and spatiotemporal databases [10, 17, 18, 20–22] inapplicable to moving objects in indoor spaces. First, the existing Euclidean and network distances do not fit indoor spaces that usually feature complex entities and topologies. Refer to the example in Figure 1. If we ignore the indoor topology, location p_1's 1st NN is p_3 since the Euclidean distance between them is the smallest. However, p_3 is not reachable from p_1 along the straight line segment between them, so p_1's true 1st NN is p_2. Also, the conventional notion of network distance does not capture the (merely) constrained movements and the obstructed distances indicated by complex indoor entities such as rooms, doors, and hallways.

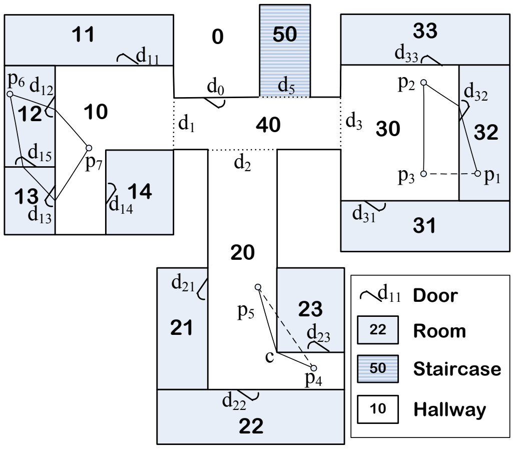


Figure 1: Example of Indoor Space, Locations, and Distances

Second, indoor positioning technologies such as RFID and Bluetooth differ fundamentally from those typically assumed in outdoor settings. Unlike GPS and cellular positioning technologies that are capable of continually reporting the position of an object quite accurately, indoor positioning technologies rely on proximity analysis [9] and are unable to report accurate or continuous locations. In particular, an indoor object is detected only when it enters the activation range of a positioning device, e.g., an RFID reader or a

EDBT 2010, March 22–26, 2010, Lausanne, Switzerland.

Bluetooth base station, which may occur infrequently. As a result, the limitations of indoor positioning create inherent uncertainty in the positioning data of indoor moving objects.

To accommodate the indoor setting, we propose the notion of minimal indoor walking distance (MIWD) as the distance metric for indoor spaces. The MIWD between two indoor locations is the shortest distance a person has to walk to reach the one from the other. In Figure 1, the MIWD between p_1 and p_2 is the length of the polyline from p_1 to p_2 via the door d_{32}.

Notably, for an indoor space, we capture all doors in a graph in which each vertex corresponds to a door and edges indicate which pairs of doors can be reached from one another without using a third door. This structure and additional mappings enable us to compute all door-to-door MIWDs efficiently. We then propose an algorithm that computes the MIWD for two arbitrary indoor locations by exploiting the pre-computed door-to-door MIWDs.

We also propose a complete set of techniques for indoor moving object management. A deployment of positioning devices partitions the indoor space into cells in the sense that all movements between cells are detected by some positioning device. We accordingly present a positioning device deployment graph in which each vertex corresponds to a cell and each edge corresponds to relevant devices. We then assign the indoor moving objects to states according to the device activation ranges and cells. The resulting states are utilized to design effective hash based indexing structures for indoor objects. The above also enable us to formalize the uncertainty of object locations.

On top of these foundations, we provide a solution for probabilistic threshold kNN (PTkNN) queries over moving objects. Given an indoor moving object set O, an indoor query location q, and a probability threshold T, a PTkNN query returns all k sized subsets of O with probability larger than T of being the kNNs of q.

We propose a combination of three techniques for the efficient processing of PTkNN queries. First, MIWDs between the query location q and indoor moving objects are used to prune objects too far away from q, which reduces the object set O to a smaller candidate set. Second, efficient probability estimation is used to prune unqualified candidate k-subsets from the candidate set. Third, the final query result is determined by efficient probability evaluation for the remaining candidate objects and k-subsets.

A comprehensive empirical study is conducted using both synthetic and real data. The results show that the provided data management foundation is effective and that the proposed PTkNN query processing is efficient and scalable.

We summarize our contributions as follows. First, we propose the minimum indoor walking distance for indoor spaces (Section 3.1). Second, we formalize the uncertainty of indoor moving object locations (Section 4.2), based on a symbolic indoor positioning and hash based object indexing scheme. Third, we provide efficient means of computing PTkNN queries over indoor moving objects (Section 5). Fourth, we conduct a comprehensive empirical study on our proposals using both synthetic and real data (Section 6). To the best of our knowledge, this paper is the first work to model indoor position uncertainty, and the first to study efficient kNN queries in indoor space.

The remainder of the paper is organized as follows. Section 2 briefly reviews related work. Section 3 presents the foundations. Section 4 elaborates on the management of indoor moving objects and the deriving of their uncertain regions. Section 5 details the techniques for efficient processing of probabilistic threshold k nearest neighbor queries over indoor moving objects. Section 6 conducts the empirical study. Finally, section 7 offers conclusions and discusses directions for future work.

2. RELATED WORK

Symbolic space models are often preferred over geometric models in the modeling of indoor space because they are better able to capture the movement-related semantics associated with indoor entities [2]. A graph-based model for indoor space is proposed to support efficient indoor tracking [12], which serves as the basis for the object management in this paper. The state definition and hash indexing method for indoor moving objects, proposed in [24] for continuous range monitoring over indoor moving objects, still form the data management foundations for snapshot kNN queries queries in this paper. Li and Lee [14] define the indoor nearest neighbors by the minimal number of doors to go through, whereas we employ a more refined general metric of minimum indoor walking distance.

kNN queries constitute fundamental functionality in spatial and spatiotemporal databases. In Euclidean spaces, spatial data are usually indexed by some spatial access method, typically an R-tree [8]. The Euclidean distances between a query point and index entry MBRs are used to facilitate pruning in R-tree based query processing, which can be conducted in either a depth-first [17] or a best-first [10] manner. kNN queries over free-moving objects come in several variants. In one variant, a query involves a moving query point and static data points indexed by an R-tree. Song and Roussopoulos [20] propose a sampling-based method to periodically reevaluate kNN queries via the R-tree. Tao et al. [22] propose an algorithm that searches the R-tree only once to find k NNs for all query positions along a line segment. Other variants support both static and moving query points over moving objects. Typically, such proposals [18, 21] search a TPR-tree [19] that indexes the moving objects. In the context of spatial networks, network distance is the metric used [11]. Different techniques are proposed for static query points [13, 16] and moving query points [4]. All these kNN proposals fall short in our setting, which requires specific modeling, indexing, and query processing.

NN and kNN queries over uncertain data have also been studied. Cheng et al. [5] propose a query that returns all objects along with their non-zero probability of being the NN of a query point. An R-tree-like index is employed to process such queries. Beskales et al. [3] propose the top-k probable nearest neighbor (Top$k$-PNN) query that returns objects with the highest probability of being the NN of a query point. The probability in a Top$k$-PNN query differs from that in our PTkNN queries. Therefore, a Top$k$-PNN query may return k objects that do not form a qualified k-subset as k NNs. Cheng et al. propose a framework [6] for evaluating probability threshold kNN over uncertain moving objects in Euclidean space, which consists of filtering, probabilistic candidate selection and verification.

Our PTkNN query definition is similar to that in [6], as both return the k-subsets with the highest probabilities of being the kNNs. The query processing framework in [6] is also adopted in this paper. However, this paper distinguishes itself with several unique characteristics. First, it uses the novel minimum indoor walking distance (MIWD) as the underlying distance metric instead of the Euclidean distance. Second, the indoor distance based pruning in this paper is able to prune not only individual objects, but also groups of objects based on the cells defined in the positioning device deployment graph. Third, this paper presents a unique formalization of the uncertainty of indoor moving object locations. Last but not least, partially due to the previous point, the probability estimation and evaluation for indoor objects in this paper are different from those for outdoor moving objects in [6].

3. INDOOR SPACE MODELING AND POSITIONING

A simplified plan of the first floor of the computer science department at Aalborg university is shown in Figure 1. The numbers are labels for rooms, hallways, and staircases. The building is divided into three clusters, each with its own hallway and rooms, that are connected by a common hallway, labeled 40. Objects can reach other floors via a staircase, labeled 50, or leave the building through the main entrance, labeled 0. For simplicity, we regard the hallways and staircases as rooms. For example, we use "room 10" for "hallway 10." Each room may have several doors, which are labeled as d_i. The connections between different clusters are also treated as a virtual door, e.g. d_1, d_2 and d_3.

Section 3.1 details our proposal of the minimal indoor walking distance. Sections 3.2 and 3.3 offers background information on indoor positioning, details of which can be found elsewhere [12]. Important notation is listed in Table 1.

Symbol	Meaning
p, q	Indoor positions
Σ_{rooms}	The set of rooms
Σ_{doors}	The set of doors
$d_{MIW}(p, q)$	Minimal indoor walking distance between p and q
O	The set of indoor moving objects
o, o_i	Indoor moving objects
$o.V_{max}$	o's maximum speed
$UR(o, t)$	o's uncertain region at time t
$C_{MSC}(o, dev, t)$	o's maximum-speed constrained circle at time t

Table 1: Notation

3.1 Minimal Indoor Walking Distance

As pointed in Section 1, the predominant Euclidean and network distances do not work well in indoor spaces. We therefore propose a new notion of *Minimal Indoor Walking Distance (MIWD)* for use as the distance metric in indoor spaces. For two positions p and q, this distance is given as $d_{MIW}(p, q)$. To facilitate computing minimal indoor walking distances, we need two mappings.

The set Σ_{rooms} contains all the rooms in the floor plan; and the set Σ_{doors} contains all the doors in the indoor space. The mapping *Rooms* determines the room of an indoor position.

$$Rooms : positions \rightarrow \Sigma_{rooms}$$

Each door connects two adjacent rooms in the sense that one can move from one room to the other through the door. The mapping *Doors* maps a room to the doors that connect the room to an adjacent room.

$$Doors : \Sigma_{rooms} \rightarrow 2^{\Sigma_{doors}}.$$

In Figure 1, $Rooms(p_1)$ returns room 32, and $Doors(room\ 12)$ returns the set $\{d_{12}, d_{15}\}$.

If p and q are in the same room, the intra-room obstructed distance [25] between the two, termed as $d_o(p, q)$, is first calculated. If no obstacles are present between p and q, $d_o(p, q)$ is equal to the Euclidean distance between the two. In Figure 1, positions p_2 and p_3 are both in room 30, and the line segment between them is fully in room 30. Therefore, the obstructed distance $d_o(p_2, p_3)$ between them is exactly $|p_2p_3|$[1].

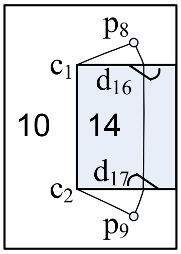


Figure 2: Example

Next, consider the indoor positions p_4 and p_5 in Figure 1. The line segment from p_4 to p_5 intersects room 23, which means that no object can move from p_4 to p_5 according to a straight line segment. In such cases, the obstructed distance accounts for the obstacles. In particular, the object needs to go from p_4 to the corner c of room 23 and then to p_5. Thus, $d_o(p_4, p_5)$ equals $|p_4c| + |cp_5|$. It is noteworthy that $d_o(p, q)$ for p and q in the same room may not be the true $d_{MIW}(p, q)$, as passing through other room(s) may result in shorter distance(s). For example, in Figure 2, the obstructed distance $d_o(p_8, q_9) = |p_8c_1| + |c_1c_2| + |c_2p_9|$ is longer than the true MIWD $|p_8d_{16}| + |d_{16}d_{17}| + |d_{17}p_9|$. Such a MIWD distance is obtained by regarding p and q as locations in different rooms, as to be deliberated next.

If two indoor positions are in different rooms, the minimal indoor walking distance should take into account the doors connecting the rooms. In Figure 1, point p_1 and point p_2 is in room 32 and room 30, respectively. The only connection between the two rooms is the door d_{32}. Therefore, $d_{MIW}(p_1, p_2) = |p_1d_{32}| + |d_{32}p_2|$.

In a more complicated situation, there may exist several paths (that go through different rooms and doors) from p to q. The correct $d_{MIW}(p, q)$ is then the distance of the shortest such paths. For example, point p_6 and point p_7 are located in room 12 and room 10, respectively, and more than one path between them exists. To reach p_7 from p_6, one can go either directly through door d_{12}, resulting in a distance $|p_6d_{12}| + |d_{12}p_7|$, or through door d_{15} followed by d_{13}, resulting in a distance $|p_6d_{15}| + |d_{15}d_{13}| + |d_{13}p_7|$. In this example, $d_{MIW}(p_6, p_7) = |p_6d_{12}| + |d_{12}p_7|$.

In order to compute $d_{MIW}(p, q)$ for any location of p and q, we need the ability to retrieve the connecting doors between two rooms. For that purpose, we define the *Doors Graph*, in which each vertex corresponds to a specific door in the indoor space. Based on the *Doors* mapping, if two doors belong to the same room, the two corresponding vertices are connected as an edge. Formally, the *Doors Graph* is defined as a weighted graph $G_d = \langle D, E, \ell_{weight} \rangle$, where:

(1) $D = \Sigma_{doors}$ is the set of vertices.

(2) E is the set of edges. An edge $\{d_i, d_j\}$ exists if a room rm exists in Σ_{rooms} such that $\{d_i, d_j\} \subseteq Doors(rm)$.

(3) ℓ_{weight}: $E \rightarrow \mathcal{R}$ assigns to an edge the obstructed distance between the two doors represented by the edge: ℓ_{weight} $(\{d_i, d_j\}) = d_o(d_i, d_j)$.

Due to complex indoor topology, a pair of doors together may belong to different rooms. Consequently, to get from one door to the other without leaving a single room, one may have different rooms to choose from. In such cases, the shortest intra-room distance is assigned to the corresponding edge as the weight. For example, in Figure 2, the weight of edge $\{d_{16}, d_{17}\}$ is assigned as the obstructed distance in room 14, i.e. $d_o(d_{16}, d_{17}) = |d_{16}d_{17}|$; but not the obstructed distance in room 10 which is $d_o(d_{16}, d_{17}) = |d_{16}c_1| + |c_1c_2| + |c_2d_{17}|$.

The doors graph corresponding to Figure 1 are shown in Figure 3. Based on this graph, all door-to-door shortest path distances can be computed and recorded in a hash table $D2D$:

$$D2D : \{(d_p, d_q)\} \rightarrow \mathcal{R}\ , \ \ d_p, d_q \in \Sigma_{doors}.$$

[1] $|p_ip_j|$ denotes the Euclidean distance between positions p_i and p_j.

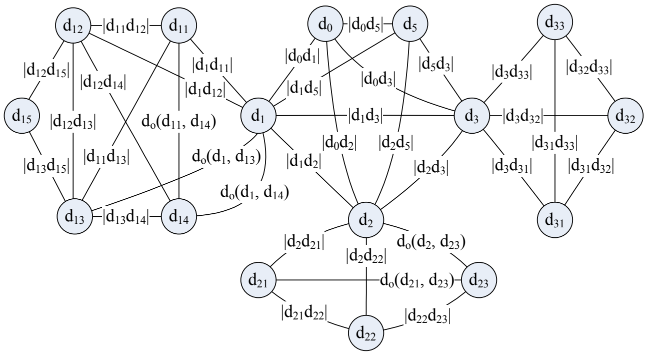


Figure 3: Doors Graph

Consequently, for two positions p and q located in different rooms, $d_{MIW}(d_p, d_q)$ is the minimum sum $d_o(p, d_p) + D2D(d_p, d_q) + d_o(d_q, q)$ where d_p ranges over all doors of room p and d_q ranges over all doors of room q.

To summarize, $d_{MIW}(p, q)$ is computed by Algorithm 1.

Algorithm 1 d_{MIW}(Position p, Position q)

```
if $Rooms(p)$=$Rooms(q)$ then
    $minDist \leftarrow d_o(p, q)$;
else
    $minDist \leftarrow +\infty$
for each door $d_p$ in $Doors(Rooms(p))$ do
    for each door $d_q \neq d_p$ in $Doors(Rooms(q))$ do
        $l \leftarrow d_o(p, d_p) + d_o(d_q, q) + D2D(d_p, d_q)$
        if $l < minDist$ then
            $minDist \leftarrow l$;
return $minDist$;
```

We note that, it is possible to adapt this notion of distance to accommodate other semantics. For example, a person might prefer a longer indoor path that, however, passes as few doors as possible. To support this, we only need to assign a uniform edge weight of 1 to each edge in the doors graph.

3.2 Symbolic Indoor Positioning

We assume the use of presence, or proximity-based, sensing technologies such as RFID or Bluetooth. We do not consider signal strength [1], as the activation ranges of RFID readers in our setting are relatively small (tens of centimeters to a few meters [23]).

These technologies employ proximity analysis [9], which determines when an object is within the activation range of a device. Each device detects and reports the observed objects at a relatively high sampling rate. A reading $(deviceID,\ objectID,\ t)$ states that object $objectID$ is detected by device $deviceID$ at time t.

A positioning device deployment is shown in Figure 1, where the numbered red circles indicate the locations and activation ranges of devices. For positioning devices with overlapping ranges, we treat the intersection as the activation range of a new, virtual positioning device. For example, the intersection of $device_1$ and $device_{1'}$ is assigned to a virtual device $device_{1'1}$. An object seen by $device_1$, but not $device_{1'}$, is then in the non-intersecting part of the range of $device_1$. Different from overlapping devices here, so-called paired devices (covered in Section 3.3) are used to detect movement direction, e.g., the entry/exit a room.

For each object, only its first and last appearances in a device's range are of interest. We thus introduce a pre-processing module in-between the sensing devices and our object management module that continuously (according to the sampling unit T_s) receives readings from all positioning devices and outputs records in of form $(deviceID,\ objectID,\ t,\ flag)$, where $flag = ENTER$ indicates the object is entering the device's activation range, and $flag = LEAVE$ indicates the object is leaving the range. The $deviceID$ can be a virtual device. Unless explicitly stated otherwise, this applies to all $deviceID$s in the rest of the paper. Due to space limitations, we omit the details of the pre-processing module.

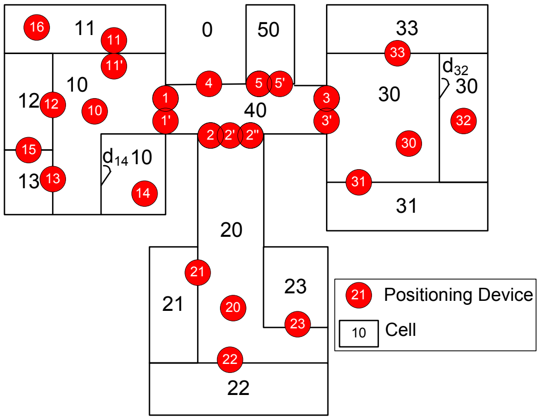


Figure 4: Positioning Device Deployment

3.3 Positioning Device Deployment Graph

In a deployment of a set of positioning devices, we differentiate between two types of devices.

Partitioning devices partition the indoor space into cells in the sense that an object cannot move from one cell to another without being observed. An example is a device deployed by the single door of a room. There are two options for partitioning devices. First, *undirected partitioning devices* (*UP*) cannot detect movement directions between cells. For example, $device_{21}$ cannot tell whether an observed object enters or leaves cell c_{21}. Note that $device_1$, $device_{1'}$, and $device_{1'1}$ are also undirected. Second, *directed partitioning devices* (*DP*) consist of entry/exit pairs of sensor that enables the movement direction of an object to be inferred by the reading sequence, e.g., $device_{11}$ and $device_{11'}$ in Figure 4.

Next, *presence devices* (*PR*) simply sense the presences of objects in their range, but do not contribute to the space partitioning. These are exemplified by $device_{10}$ in Figure 4.

To facilitate query processing, three mappings are defined. A *Devices* mapping structure maintains the activation range, intersecting room, and type of each positioning device:

$$Devices:\ \ \Sigma_{devices} \rightarrow \{(AR, \Sigma_{rooms}, TYPE)\}.$$

Here, $\Sigma_{devices}$ is the set of all devices; AR is the set of activation ranges (usually a range is a circular region); Σ_{rooms} captures the set of rooms that intersect with AR; and $TYPE = \{UP, DP, PR\}$ is the set of device types.

In the $PA2D$ mapping, each device dev is mapped to the door that is covered by its activation range:

$$PA2D : \Sigma_{devices} \rightarrow \Sigma_{doors}.$$

For example, $device_{12}$ in Figure 4 is mapped to door d_{12}.

For each presence device dev, the distances from its location to all the doors of the room in which it is deployed are recorded into the $PR2D$ hash table:

$$PR2D : \{(dev, d)\} \rightarrow \mathcal{R}\ ,\ dev \in \Sigma_{devices}, d \in \Sigma_{doors}.$$

For example, presence device $device_{10}$ is deployed in room 10 in Figure 4, and the distance from its location of $device_{10}$ to doors d_1, d_{11}, d_{12}, d_{13}, and d_{14} (see Figure 1) are recorded in $PR2D$.

Finally, to facilitate the tracking and querying of moving objects, a deployment graph is created based on the topological relationship of the floor plan and the positioning device deployment. Formally, a deployment graph is a labeled graph $G = \langle C, E, \Sigma_{devices}, \ell_E \rangle$, where:

(1) C is the vertex set. Each vertex corresponds to a cell.

(2) E is the edge set consisting of unordered pairs of vertices. An edge indicates that its two cells are connected.

(3) ℓ_E: $E \rightarrow 2^{\Sigma_{devices}}$ assigns a set of positioning devices to an edge. Specifically, a non-loop edge is labeled by the partitioning device(s) partitioning the two cells, and a loop edge captures the presence devices in a cell.

Using an existing deployment graph construction algorithm [12], the deployment graph corresponding to Figure 4 is shown in Figure 5, where label D_i indicates a positioning device $device_i$.

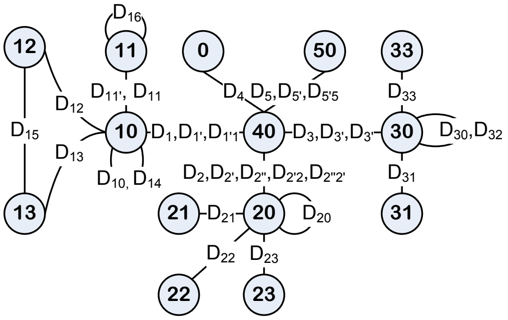


Figure 5: Positioning Device Deployment Graph

Each cell created by a device deployment corresponds to one or more rooms in the floor plan. For example, cell c_{10} corresponds to rooms 14 and 10 because an object can go from room 14 to room 10 without being observed by a positioning device. Similarly, cell c_{30} corresponds to rooms 32 and 30. The following *Cells* mapping maintains the corresponding relationship.

$$Cells : C \rightarrow 2^{\Sigma_{rooms}}.$$

Also, a room or a long hallway may be divided into cells by a deployment of partitioning devices. The resulting divisions can be regarded as virtual doors, which should then be reflected in the *Doors* graph and the $D2D$ mapping.

The rooms in a floor plan partition the floor plan in a way independent of a particular deployment of positioning devices. In contrast, a cell partitioning is caused by a deployment of positioning devices. The extent of a cell is the union of the extents of the rooms that make up the cell, excluding the ranges of intersecting devices. In the running example, cell c_{10} is the union of rooms 10 and 14 excluding the activation ranges of $device_1$, $device_{1'}$, $device_{1'1}$ $device_{10}$, $device_{11'}$, $device_{12}$, and $device_{13}$. Thus an indoor space is partitioned into activation ranges and cells.

4. UNCERTAINTY OF INDOOR MOVING OBJECTS

Section 4.1 categorizes indoor moving objects according to their positions, and it indexes them using hashing structures. A preliminary version of this can be found elsewhere [24]. We include it here to render the presentation self-contained. Section 4.2 derives the uncertain region that an indoor moving object can belong to at a given time.

4.1 Management of Indoor Moving Objects

Given a deployment of indoor positioning devices, an indoor space is partitioned into an *active subspace* and an *inactive subspace*. The active subspace is the union of the activation ranges of all positioning devices, and it usually consists of disconnected sub-regions. The inactive subspace is the part of space that is not covered by any positioning device.

If an object is in the active space, it must be in some device's activation range. With the $Devices$ mapping (see Section 3.3), we are able to directly determine the object's whereabouts. If an object is in the inactive space, additional processing and information is needed to infer its possible locations.

Thus, an object is *active* or *inactive*, depending on its subspace membership. Inactive objects may be in a *deterministic state* and a *nondeterministic state*. The deterministic state indicates that an object's current location is guaranteed to be in only one specific cell (as defined in Section 3.3), and the nondeterministic state indicates that the object's current location may be in one of several cells.

More specifically, if a moving object leaves (the activation range of) a presence device d, it must be still in the cell $G.\ell_E^{-1}(d)$ before it is again detected [2]. Therefore, its state changes from active to deterministic. In Figures 4 and 5, if an object leaves $device_{10}$, it must enter c_{10}. If an object leaves a directed partitioning device, the cell the object is entering can be determined from the reading sequence. Therefore, its state also changes from active to deterministic. In the running example, if an object is seen at $device_{11'}$ and then $device_{11}$, it must enter c_{11}.

In contrast, if an object leaves a undirected partitioning device, the object can be in any of the cell in $G.\ell_E^{-1}(d)$. Therefore, its state changes from active to nondeterministic. In the running example, if a moving object leaves $device_{12}$, it can be in either c_{10} or c_{12}.

On the other hand, if a moving object enters (the activation range of) any positioning device, its state changes from inactive (deterministic or nondeterministic) to active.

The resulting state transition diagram is shown in Figure 6. Based on it, we employ an indexing scheme that utilizes several hash tables. Let O be the set of all the moving objects in the indoor space. For positioning devices, a *Device Hash Table (DHT)* is created that maps a given positioning device to the set of active objects in the device's activation range: $DHT : \Sigma_{devices} \rightarrow 2^O$.

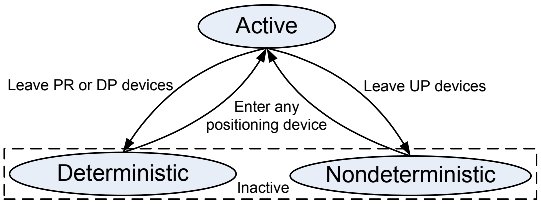


Figure 6: Moving Object State Transition Diagram

[2] $G.\ell_E^{-1}$ is the reverse function of $G.\ell_E$ introduced in Section 3.3. For simplicity, we use $G.\ell_E^{-1}(d)$ to denote $G.\ell_E^{-1}(D)$, where $D \subseteq \Sigma_{devices}$. Specifically, $D = \{d\}$ if d is a non-overlapping UP device or the only PR in a cell; D is the set of overlapping UP devices if d is one of them; D is the set of two DP devices if d is one of them; otherwise, D is the set of all PR devices in the same cell as d. Note that for an arbitrary device d, the corresponding set D is unique.

Two hash tables are maintained for the cells. A *Cell Deterministic Hash Table (CDHT)* maps a cell to the set of deterministic objects in it: $CDHT : C \rightarrow 2^O$. Next, a *Cell Nondeterministic Hash Table (CNHT)* maps a cell to the set of nondeterministic objects in it: $CNHT : C \rightarrow 2^O$.

In addition, the *Object Hash Table (OHT)* and *Object Leave Hash Table (OLHT)* are maintained for all objects. OHT maps an object identifier to the corresponding object state tuple: $OHT : O \rightarrow \{(STATE,\ t,\ IDSet)\}$. Here $STATE$ denotes the object's current state; t is the start timestamp of the state; $IDSet$ is a set of cell identifiers or a set of device identifiers, indicating where the object can currently be. If the object's state is active, $IDSet$ is a singleton set of the corresponding device identifier. If the state is deterministic, $IDSet$ is a singleton set of the corresponding cell identifier. If the state is nondeterministic, $IDSet$ is a set of identifiers of all the cells in which the object can currently be.

$OLHT$ maps an object identifier to its last $LEAVE$ observation, which is designed to facilitate the uncertainty region determination (to be discussed in detail in Section 4.2): $OLHT : O \rightarrow \{(deviceID,\ t)\}$.

These hash tables need updating whenever there is a new output from the pre-processing module. The update algorithm, described in Algorithm 2, handles a record received from the pre-processing module according to its $flag$ value.

Algorithm 2 UpdateHashTables(Pre-processing output S, DeploymentGraph G)

```
 1: IDSet $sSet \leftarrow \emptyset$;
 2: **if** $S.flag$ = $ENTER$ **then**
 3:     $sSet \leftarrow OHT[S.objectID].IDSet$;
 4:     **if** $OHT[S.objectID].STATE$ = $Active$ **then**
 5:         **for** the single element $c$ in $sSet$ **do**
 6:             Delete $S.objectID$ from $DHT[c]$;
 7:     **else if** $OHT[S.objectID].STATE$ = $Deterministic$ **then**
 8:         **for** the single element $c$ in $sSet$ **do**
 9:             Delete $S.objectID$ from $CDHT[c]$;
10:     **else**
11:         **for** each element $c$ in $sSet$ **do**
12:             Delete $S.objectID$ from $CNHT[c]$;
13:     Add $S.objectID$ to $DHT[S.deviceID]$;
14:     $OHT[S.objectID] \leftarrow (Active,\ S.t,\ \{S.deviceID\})$;
15: **else**
16:     Delete $S.objectID$ from $DHT[S.deviceID]$;
17:     $sSet \leftarrow G.\ell_E^{-1}(S.deviceID)$;
18:     **if** $Devices(S.deviceID).TYPE$ = UP **then**
19:         $OHT[S.objectID] \leftarrow (Nondeterministic, S.t, sSet)$;
20:         **for** each element $c$ in $sSet$ **do**
21:             Add $S.objectID$ to $CNHT[c]$;
22:     **else**
23:         $OHT[S.objectID] \leftarrow (Deterministic, S.t, sSet)$;
24:         **for** the single element $c$ in $sSet$ **do**
25:             Add $S.objectID$ to $CDHT[c]$;
26:     $OLHT[S.objectID] \leftarrow (S.deviceID, S.t)$;
```

For an $ENTER$ record, if the object's previous state is active, it is deleted from the corresponding device's DHT (lines 4–6). If its previous state is deterministic, it is deleted from the corresponding cell's $CDHT$ (lines 7–9). Otherwise, its previous state is nondeterministic, and it is deleted from all corresponding cells' $CNHT$s (lines 10–12). After the deletion, the object is added into the DHT of the current device, and its state is updated accordingly (lines 13–14).

For a $LEAVE$ record, the object is deleted from the corresponding device's DHT (lines 15–16). The possible cells are determined by the function $G.\ell_E^{-1}$ (lines 17). If the object leaves an UP device, its state is set to nondeterministic, and the object is added into all the corresponding cells' $CNHT$s (lines 18–21). If the object leaves a DP or PR device, its state is set to deterministic, and the object is added into the corresponding cell's $CDHT$ (lines 22–25). At last, the corresponding entry in $OLHT$ is updated (line 26).

4.2 Deriving Uncertain Regions for Indoor Moving Objects

We capture the *Uncertainty Region (UR)* of an indoor object at the time a query is issued. As for outdoor moving objects [5], the uncertainty region of an indoor object o at time t, denoted by $UR(o, t)$, is a region such that o must be in this region at time t.

In general terms, the location of an object o_i can be modeled as a random variable with a probability density function $f_{o_i}(x, y, t)$ that has non-zero values only in o_i's uncertainty region $UR(o_i, t)$ and for which $\int_{UR(o_i,t)} f_{o_i}(x, y, t)dxdy = 1$.

Indoor objects have more constraints on their movements than have free-moving outdoor objects. For example, if an object's destination is not in its current room, the object must pass through one or more doors to reach its destination. Because they do not capture the indoor topologies and the associated constraints and obstacles, uncertainty models [15] for outdoor objects do not apply well in our indoor setting.

We thus proceed to present an uncertainty model designed for indoor moving objects. In the following discussion, we assume that an object has the same probability to be located anywhere inside its uncertainty region. That is, the probability is distributed uniformly in the object's uncertainty region:

$$f_{o_i}(x, y, t) = \frac{1}{Area(UR(o_i, t))},\ (x, y) \in UR(o_i, t).$$

According to the analysis on the states of indoor moving objects in Section 4.1, the uncertainty regions of indoor moving objects can be characterized as follows. The uncertainty region of an active object is the activation range of the corresponding device, and the uncertainty region of an inactive object is the cell or cells that the object can belong to.

If the object's maximum speed V_{max} is given, its uncertainty region can be captured at a finer granularity. The uncertainty region of a deterministic object is refined as the intersection between the object's cell and its *maximum-speed constrained circle*. For a nondeterministic object, the region is the union of the intersection between each cell and the circle.

Let the last $LEAVE$ observation of object o be from device dev at time t and let the time duration from t to the current time be $\Delta t = t_{now} - t$. Assuming that the object o moves in a straight line, the longest possible distance o can move away from the boundary of dev's activation range is $o.V_{max} \cdot \Delta t$. Formally, the maximum-speed constrained circle $C_{MSC}(o, dev, t)$ of o is defined as the circle centered at dev's deployment location and with radius $o.V_{max} \cdot \Delta t$ plus the radius of dev's activation range. We also exclude the activation range of dev from the circle.

Consider Figure 7 and assume that object o left $device_{16}$ at time t. Its maximum-speed constrained circle $C_{MSC}(o, device_{16}, t)$ is then indicated by R_1 in the figure. Since $device_{16}$ is a presence device, after leaving $device_{16}$ the inactive object o must be in the cell c_{11} (according to $G.\ell_E^{-1}(device_{16})$). Due to the two constraints, object o's uncertainty region is the intersection of cell c_{11} and circle R_1, i.e., the shaded region in the top-left part of Figure 7.

If the cell where the deterministic object resides has more than one room, e.g., the cell c_{10} contains room 10 and room 14, the determination of uncertainty region is more complicated. Suppose object o left $device_{10}$ at timestamp t. According to $G.\ell_E^{-1}(device_{10})$, o should be in cell c_{10} after leaving $device_{10}$. From the $Devices$

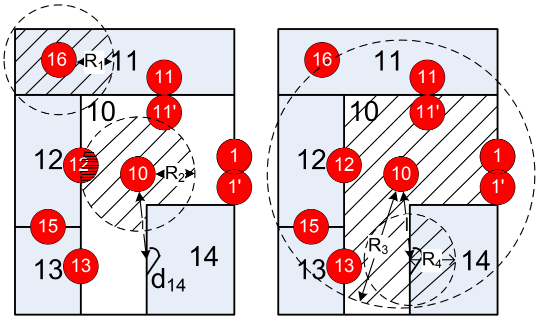


Figure 7: Uncertainty Model for Inactive Objects

and $PR2D$ mappings, we can figure out that the $device_{10}$ resides in room 10, and the distance from $device_{10}$ to the door d_{14} is l. If the maximum speed constraint does not guarantee that o can have gone through the door d_{14}, i.e. $o.V_{max} \cdot (t_{now} - t) < l$, the object o must be in the room 10. Thus, the uncertainty region is the intersection between room 10 and $C_{MSC}(o, device_{10}, t)$, which is indicated by R_2 to the left of Figure 7.

On the other hand, referring to the right part of Figure 7, if $o.V_{max} \cdot (t_{now} - t) \geq l$, the object o may have entered room 14. Its uncertainty region therefore contains two parts: the intersection between room 10 and $C_{MSC}(o, device_{10}, t)$ (indicated by R_3); the intersection between room 14 and the circle with door d_{14} as the center and $R_4 = o.V_{max} \cdot (t_{now} - t - l/o.V_{max})$ as the radius.

The uncertainty region of an active object can be also refined as the intersection of the activation range of the corresponding device and the object's maximum-speed constrained circle. Using the running example, suppose object o left $device_{10}$ at time t and that o is then observed by $device_{12}$. The uncertainty region of o is the intersection of the activation range of $device_{12}$ and $C_{MSC}(o, device_{10}, t)$, which is shown within the activation range of $device_{12}$ in Figure 7.

Algorithm 3 computes the uncertainty region of an object o at the current time t_{now}. Note that o's last $LEAVE$ observation can be obtained from hash table $OLHT$, from which the corresponding device dev and timestamp t are also found (lines 4–5). If object o is an active object, the activation range re of the corresponding device is obtained from the $Devices$ mapping. Then o's uncertainty region is the intersection of re and the maximum-speed constrained circle $C_{MSC}(o, dev, t)$ (lines 6–8).

Otherwise, the possible cells in which the object may reside are determined from the deployment graph (line 10). For each possible cell, all its corresponding rooms are determined from the $Cells$ mapping (line 11). For each possible room rm, if the device dev is deployed in it, o's uncertainty region is the intersection of the room and $C_{MSC}(o, dev, t)$ (lines 12–13). If not, the door d between room rm and the room in which the device dev is deployed is determined. The distance f from the device dev to the door d is computed from $PR2D$ or $D2D$ according to device dev's type. The uncertainty region is the intersection of room rm and the circle with d as center and $radius$ as radius (lines 14–22).

5. PT kNN QUERY PROCESSING

5.1 Definition and Overview

The query under consideration is defined as follows.

DEFINITION 1. ***(Indoor Probabilistic Threshold*** k***NN Query)*** *Given a set of indoor moving objects* O $=\{o_1, o_2, ..., o_n\}$ *and a threshold value* $T \in (0, 1]$, *a PT*k*NN query issued at time* t *with query location* q *returns a result set* $\mathbf{R} = \{A \mid A \subseteq O \wedge |A| = k \wedge prob(A) > T\}$, *where* $prob(A)$ *is the probability that* A *contains the* k *nearest neighbors of the query location* q *at time* t.

Algorithm 3 UR(Object o, DeploymentGraph G)

```
1:  Region $UR \leftarrow \emptyset$;
2:  Door $d \leftarrow \emptyset$;
3:  Integer $radius \leftarrow 0$;
4:  Device $dev \leftarrow OLHT[o].deviceID$;
5:  TimeStamp $t \leftarrow OLHT[o].t$;
6:  if $OHT[o].STATE$=$Active$ then
7:      Region $re \leftarrow Devices(OHT[o].IDSet).AR$;
8:      $UR \leftarrow re \cap C_{MSC}(o, dev, t)$;
9:  else
10:     for each cell $c$ in $G.\ell_E^{-1}(dev)$ do
11:         for each room $rm$ in $Cells(c)$ do
12:             if $rm$ in $Devices(dev).RoomSet$ then
13:                 $UR \leftarrow UR \cup (rm \cap C_{MSC}(o, dev, t))$;
14:             else
15:                 Room $rm2 \leftarrow Cells(c) \cap Devices(dev).RoomSet$;
16:                 $d \leftarrow Doors(rm2) \cap Doors(rm)$;
17:                 if $Devices(dev).TYPE$ = $PR$ then
18:                     $radius \leftarrow o.V_{max} \cdot (t_{now} - t) - PR2D(dev, d)$;
19:                 else
20:                     Door $d' \leftarrow PA2D(dev)$;
21:                     $radius \leftarrow o.V_{max} \cdot (t_{now} - t) - D2D(d, d')$;
22:                 $UR \leftarrow UR \cup (rm \cap$ Circle($d$, $radius$));
23: return $UR$;
```

The definition of $prob(A)$ will be formalized in Section 5.3. Consider the four moving objects in Figure 8. Object o_1 is being observed by $device_{21}$, and its uncertainty region is the activation range of that device. Object o_2, o_3, and o_4 recently left $device_{20}$, $device_{2'}$, and $device_{33}$, respectively. According to the discussion in Section 4.2, their uncertainty regions are captured by three maximum-speed constrained circles: $C_{MSC}(o_2, device_{20}, t)$, $C_{MSC}(o_3, device_{2'}, t)$, and $C_{MSC}(o_4, device_{33}, t)$, shown as solid circles excluding any activation ranges in Figure 8.

Assuming a 2NN query issued at time t with location q, $\binom{4}{2} =$ 6 2-subsets can be in the result set. When the number of moving objects in the indoor space increases, the number of k-subsets (A in Definition 1) in the result set $\mathbf{R}$ will increase exponentially. Specifically, there are $\binom{n}{k}$ possible k-subsets for a PTkNN query over n objects. Accordingly, computing the probability $prob(A)$ for each k-subset A will incur considerable computation cost and thus result in very slow query response.

We propose three techniques that speed up PTkNN query processing. First, minimum indoor walking distances between the query location and the (uncertainty regions of) objects are used to prune the objects too far away to be in any possible k-subset A (Section 5.2), which usually results in a much reduced object subset $O' \subseteq O$. Second, for all k-subsets of O', cost-efficient probability estimates are used to prune the k-subsets whose probabilities definitely are lower than the specified threshold T (Section 5.3). Third, for each remaining k-subset A, $prob(A)$ is evaluated efficiently, and A is added to $\mathbf{R}$ only if $prob(A) > T$ (Section 5.4).

5.2 Indoor Distance Based Pruning

The exact MIWD from a query location q to an object o_i, i.e., $d_{MIW}(q, o_i)$, is not known because the location of o_i is described by an uncertainty region $UR(o_i, t)$ at time t. Instead, we define the

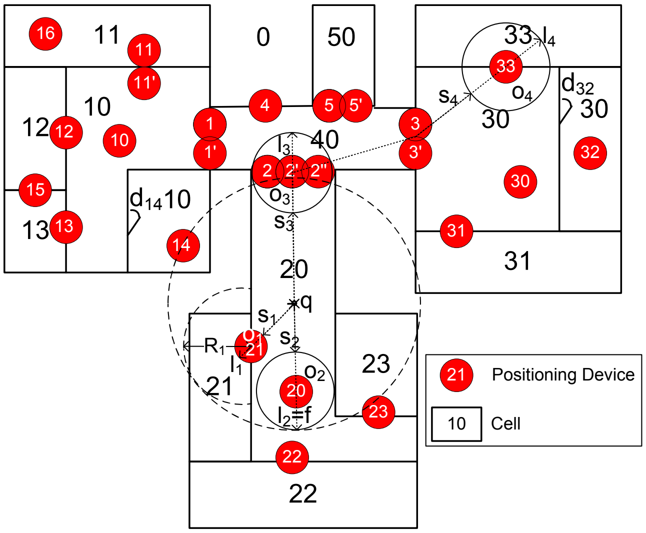


Figure 8: Indoor kNN Query Processing

minimum and maximum MIWD between q and o_i. Let s_i (l_i) be the minimum (maximum) MIWD from q to the uncertainty region of o_i:

$$s_i = \min_{p \in UR(o_i,t)} d_{MIW}(q,p), \quad l_i = \max_{p \in UR(o_i,t)} d_{MIW}(q,p).$$

If the uncertainty region $UR(o_i, t)$ is in the same room as the query location q, s_i and l_i can be obtained based on the obstructed distance. If $UR(o_i, t)$ is a circle, s_i and l_i can be determined by using the line passing the circle's center and the query location q (or the obstacle nearest to the center). For example, the minimum (maximum) distance from q to o_2 is shown as s_2 (l_2) in Figure 8. If $UR(o_i, t)$ is a polygon, all its sides and vertices need to be checked. As a special case, if the query location q is inside the uncertainty region, s_i is 0.

If the uncertainty region $UR(o_i, t)$ is not in the same room as query location q, the doors connecting the different rooms need to be considered when computing the MIWD. For object o_4 in Figure 8, doors d_2 and d_3 are considered. The s_i and l_i for the four objects are shown in Figure 8.

Let f be the k'th minimal one of all objects' l_is. If s_i of object o_i is greater than f, object o_i has no chance to be in any k-subset of the result set $\mathbf{R}$ because k objects exist that are definitely closer to q than o_i. This f value is called the *k-bound* [6].

Using the *k-bound*, some objects can be pruned early. Consider the 2NN query in Figure 8. The order among all l_is is $l_1 < l_2 < l_3 < l_4$, so $f = l_2$. As $s_4 > f$, object o_4 can be pruned safely. This pruning yields a potentially much reduced candidate object set O' ($\subseteq O$) to be considered for the given PTkNN query.

Two important observations can be used to conduct distance based pruning even more efficiently. First, the *k-bound* can be calculated and updated dynamically during the distance based pruning. The initial *k-bound* f can be obtained as soon as k objects have been seen from O. When new objects from O are being processed, f helps prune unqualified objects, and whenever possible, f is updated to a smaller value for better subsequent pruning.

Second, we do not have to determine the exact uncertainty region and calculate the exact s_i (i_i) for each object during distance based pruning. By taking advantage of the indoor space distance definition and object positioning (detailed in Section 3), we can reduce the computation cost.

This also allows us to prune objects together, based on the cells that result from the deployment of indoor positioning devices. The basic idea is this: Given a cell $cell$, if $\min_{p \in cell}\{d_{MIW}(q,p)\}$ is greater than the current *k-bound* f, all the objects currently in the cell can be safely pruned. In Figure 8, after processing objects o_1 and o_2, the current *k-bound* f is l_2, and $\min_{p \in c_{30}}\{d_{MIW}(q,p)\} = d_{MIW}(q, d_3)$ as door d_3 is the only door to c_{30}. Since $d_{MIW}(q, d_3) > f$, for any object o_i in c_{30} we have $d_{MIW}(q, o_i) > f$. Any such object o_i can be safely pruned without further processing. In this example, there is no need to compute the uncertainty region of o_4 (or any other object in c_{30}).

The distance-based pruning is described in Algorithm 4. First, the candidate object set O' and the k-bound f are initialized as empty and infinity, respectively (lines 1–2). A cell set $seeds$ records the cells we have examined, which is initialized as empty (line 3). Also, a min-heap $H\langle\langle d, v\rangle\rangle$ (line 4) gives priority to doors closer to the query location q, thus controlling the access order of relevant doors during the distance-based pruning. Note H enqueues the $\langle d, d_{MIW}(d, q)\rangle$ pair for each involved door d (line 4 in Algorithm 5).

If the query location q is in a device dev's activation range, the active objects in dev are added to the candidate set O', and the corresponding cells obtained through $G.\ell_E^{-1}(dev)$ are added to the cell set $seeds$ (lines 5–7). For each cell c obtained, both deterministic and nondeterministic objects in c are added to O', and function $EnheapDoors$ (see Algorithm 5) is called to push all the doors in c onto the min-heap H (lines 7–8).

Algorithm 4 DistancePruning(Position q, int k)

```
1:  ObjectSet $O' \leftarrow \emptyset$;
2:  Double $f \leftarrow +\infty$;
3:  CellSet $seeds \leftarrow \emptyset$;
4:  Initialize a min-heap $H\langle\langle d, v\rangle\rangle$;
5:  if $q$ is in the activation range of a device $dev$ then
6:      $O' \leftarrow DHT[dev]$;  $seeds \leftarrow G.\ell_E^{-1}(dev)$;
7:      for each cell $c$ in $seeds$ do
8:          $O' \leftarrow O' \cup CDHT[c] \cup CNHT[c]$;  $EnheapDoors(H, c)$;
9:  else
10:     Room $r \leftarrow Rooms(q)$;
11:     Cell $c \leftarrow Cells^{-1}(r)$;
12:     $O' \leftarrow CDHT[c] \cup CNHT[c]$;
13:     Add $c$ into $seeds$;  $EnheapDoors(H, c)$;
14: if $|O'| \geq k$ then
15:     $f \leftarrow Bound(O')$;
16: while $H$ is not empty do
17:     $e \leftarrow deheap(H)$;
18:     if $e.v > f$ then
19:         break;
20:     Set $e.d$ as visited;
21:     $dev \leftarrow PA2D^{-1}(e.d)$;  $O' \leftarrow O' \cup DHT[dev]$;
22:     for each cell $c$ in $G.\ell_E^{-1}(dev)$ do
23:         if $c \notin seeds$ then
24:             $O' \leftarrow O' \cup CDHT[c] \cup CNHT[c]$;
25:             for each $dev$ in $G.\ell_E(\{c, c\})$ do
26:                 if $(PR2D(dev, d) + e.v) \leq f$ then
27:                     $O' \leftarrow O' \cup UDHT[dev]$;
28:             Add $c$ into $seeds$;  $EnheapDoors(H, c)$;
29:     if $|O'| \geq k$ then
30:         $f \leftarrow Bound(O')$;
```

Otherwise, q is not in any activation range, and it must be in some cell c. Both the deterministic and nondeterministic objects in c are added to the candidate set O'. The cell c is added to the cell set $seeds$, and all its doors are pushed onto H by calling $EnheapDoors$ (lines 10–13). Note that $Cells^{-1}$ (line 11) is the reverse function of $Cells$, defined in Section 3.3, which maps a room r to the cell covering r.

If the current candidate set O' has at least k objects, function $Bound$ (see Algorithm 6) is called (lines 14–15). It determines the current k-bound f using all candidate objects, and it prunes unqualified ones according to f.

In the sequel, we need to expand to further away cells or activation ranges via doors, as an object must go through a door to reach another cell. Following the philosophy of Dijkstra's algorithm [7], the expansion is controlled by the min-heap H that stores the distances from query location q to all relevant doors.

At the beginning of each expansion step, the first entry e from H is deheaped (line 17). The expansion stops if the current door $e.d$ being processed is too far away (lines 18–19); otherwise, $e.d$ is set as visited to avoid duplicate visits (line 20). If the door $e.d$ is covered by a device dev's activation range, its active objects are added to O' (line 21).

For each cell c of the corresponding cells in $G.\ell_E^{-1}(dev)$, if c is not in $seeds$, both the deterministic and nondeterministic objects in c are added to O' (lines 22–24). For each presence device in cell c, if its indoor distance to q is smaller than f, all active objects in c are also added to O' (lines 25–27). After cell c is processed, it is added to $seeds$, and its doors are enheaped by calling function $EnheapDoors$ (line 28). At the end of each expansion step, if the current candidate set O' has at least k objects, function $Bound$ is called again to dynamically update f and reduce the candidate set O' (lines 29–30).

Algorithm 5 EnheapDoors(Heap H, Cell c)

```
for each room $r$ in $Cells(c)$ do
    for each door $d$ in $Doors(r)$ do
        if $d$ is not visited then
            enqueue($H$, $\langle d, d_{MIW}(d,q) \rangle$);
```

Algorithm 6 Bound(ObjectSet O')

```
$f \longleftarrow$ the $k$'th smallest element in $\{l_i \mid o_i \in O'\}$;
for each object $o_i$ in $O'$ do
    if $s_i < f$ then
        Delete $o_i$ from $O'$;
return $f$;
```

We regard heap insertions and deletions as characteristic operations. The worst-case time complexity of Algorithm 4 is $2 \cdot |\Sigma_{doors}|$ because each door is inserted once and deleted once.

5.3 Probability Threshold Based Pruning

After the distance based pruning, a possibly smaller candidate object set O' of k or more objects is obtained. There can still be $\binom{|O'|}{k}$ possible k-subsets in the result set $\mathbf{R}$. We proceed to prune both unqualified objects in O' and unqualified k-subsets in $\mathbf{R}$, by making use of fast probability estimates and the given probability threshold T. We assume that the distributions of all indoor moving objects are independent on each other. While, objects may move inter-dependently in some scenarios. However, determining such dependencies is a hard task that may involve large amounts of historical data. How to exploit dependencies for better performance is beyond the scope of this paper and it is an interesting future research direction.

Given an object o_i, let $P_{o_i}(r)$ be the cumulative distribution function (cdf) that o_i's MIWD to the query location q is r. In other words, $P_{o_i}(r) = Pr(d_{MIW}(q, o_i) \leq r)$. Let A be a k-subset of O'. The probability $prob(A)$ that A contains the k nearest neighbors of q satisfies:

$$prob(A) \leq \prod_{o_i \in A} P_{o_i}(f).$$

This is because only those objects within the k-bound f can be among the k nearest neighbors of q.

If $P_{o_i}(f)$ is less than the threshold T, any k-subset A that contains o_i satisfies:

$$prob(A) \leq \prod_{o_j \in A} P_{o_j}(f) \leq P_{o_i}(f) < T.$$

This means that A cannot satisfy the probability threshold T. Therefore, if $P_{o_i}(f) < T$, o_i can be safely pruned from the candidate object set O'.

All those locations with MIWD to query location q no greater than r are constrained by a *bounding region* $BR_q(r)$, which is usually composed of several intersections of rooms and circular regions. Formally, $BR_q(r)$ is defined as $Rooms(q) \cap Circle(q,r) \bigcup \cup_{rm_i \in R_r} rm_i \cap Circle(q, r'_i)$, where R_r is a set of rooms. Any room rm_i in R_r satisfies the condition that the MIWD l_i from its door to the query location q is smaller than r, and the corresponding r'_i equals r-l_i. For example, the bounding region $BR_q(f)$ in Figure 8 is indicated by two dashed circular regions: $BR_q(f) = (room_{20} \cap Circle(q, f)) \cup (room_{21} \cap Circle(d_{21}, f - |qd_{21}|))$.

Based on the bounding region $BR_q(r)$, the $P_{o_i}(r)$ can be evaluated using the following equation:

$$P_{o_i}(r) = \frac{Area(UR(o_i, t) \ \cap \ BR_q(r))}{Area(UR(o_i, t))} \quad (1)$$

In Figure 8, $P_{o_3}(f) = 0.5$. If the specified threshold $T > 0.5$, o_3 can be safely eliminated from O'.

The pseudo code of the probability threshold based pruning is given in Algorithm 7. For each object o_i in O', if its probability $P_{o_i}(f)$ is less than the threshold T, the object is removed from O' (lines 1–3). Next, we generate all possible i-subsets step by step (lines 4–15). Each i-subset A is the union of an $(i-1)$-subset B in $\mathbf{R}$ and a singleton set $\{c\}$. In particular, $\{c\}$ is not in B but is in another $(i-1)$-subset in $\mathbf{R}$. All i-subsets in $SubSet$ are obtained from all such combinations of B and $\{c\}$ based on $\mathbf{R}$. The i-subset is included in the temporary result set $\mathbf{R}$ only if the product of all its members' probabilities is greater than the threshold T (lines 16–18). This way, some unqualified i-subsets are eliminated without probability estimates. For example, if a 2-subset $\{o_1, o_2\}$ cannot satisfy the probability threshold, any i-subset (where $i > 2$) which contains $\{o_1, o_2\}$ cannot satisfy the probability threshold either.

We regard the calculation of $P_{o_i}(f)$ as the characteristic operation. If the calculated probabilities cannot be kept in memory and every probability has to be recalculated on-the-fly each time it is needed, the worst-case time complexity of Algorithm 7 is $|O'| + \sum_{i=2}^{k} \binom{|O'|}{i} \cdot i = \mathcal{O}(k \cdot 2^{|O'|})$. With enough memory for holding at least $|O'|$ double values, the worst-case time complexity is $|O'|$ because every calculated probability can be reused.

5.4 Probability Evaluation

After the probability threshold based pruning, each k-subset A in $\mathbf{R}$ may have a probability $prob(A)$ greater than the threshold T. We next present a technique to evaluate those probabilities efficiently.

Formally, the probability $prob(A)$ that k-subset A ($A \in \mathbf{R}$) contains the k nearest neighbors of the query location q is defined as

Algorithm 7 ProbTPruning(ObjectSet O', double T)

```
for each object $o_i \in O'$ do
    if $P_{o_i}(f) < T$ then
        Remove $o_i$ from $O'$;
ResultSet $\mathbf{R} \leftarrow \emptyset$;
for each object $o_i \in O'$ do
    $\mathbf{R} \leftarrow \mathbf{R} \cup \{\{o_i\}\}$;
for $i \leftarrow 2$ to $k$ do
    $SubSet \leftarrow \emptyset$;
    ObjectSet $RR \leftarrow \emptyset$;
    for each $(i-1)$-subset $B$ in $R$ do
        $RR \leftarrow RR \cup B$;
    for each $(i-1)$-subset $B$ in $R$ do
        for each object $c$ in $RR \setminus B$ do
            Add $\{B \cup c\}$ into $SubSet$;
    $\mathbf{R} \leftarrow \emptyset$;
    for each $i$-subset $A$ in $SubSet$ do
        if $\prod_{o_i \in A} P_{o_i}(f) \geq T$ then
            $\mathbf{R} \leftarrow \mathbf{R} \cup \{A\}$;
return $\mathbf{R}$;
```

follows:

$$prob(A) = \sum_{o_z \in A} \int_0^{+\infty} p_{o_z}(r) \prod_{o_i \in A \setminus \{o_z\}} P_{o_i}(r) \prod_{o_j \in O' \setminus A} (1 - P_{o_j}(r))\, dr$$

Here, $p_{o_z}(r)$ is the pdf that the distance from o_z to query point q is r; $\prod_{o_i \in A \setminus \{o_z\}} P_{o_i}(r)$ indicates the probability that all objects in A excluding o_z are within r of q; $\prod_{o_j \in O' \setminus A}(1-P_{o_j}(r))$ indicates the probability that all objects not in A are further away from q than r. For each object $o_z \in A$, the integral calculates the probability that o_z is the k-th nearest neighbor of q and that all the remaining ones in A are the first to the $(k-1)$-th nearest neighbors. As a result, the summation over all objects in A is the desired probability.

To evaluate the probabilities efficiently, we use a partition based approximate evaluation method. Let an array sl record all the minimum distances s_i in O' and the maximum distances l_i that satisfy $l_i \leq f$. This array should have g (g=$|O'| + k$) elements. We sort sl in ascending order.

Using the array sl, the bounding region $BR_q(f)$ can be partitioned into $g-1$ partitions. In particular, partition PA_x is the contour between $BR_q(sl[x-1])$ and $BR_q(sl[x])$. For Figure 8, we have $O' = \{o_1, o_2, o_4\}$ after distance based pruning and probability threshold based pruning, and therefore $g = 3+2 = 5$. Four partitions are created: $PA_1[BR_q(s_1), BR_q(s_2)]$, $PA_2[BR_q(s_2), BR_q(l_1)]$, $PA_3[BR_q(l_1), BR_q(s_3)]$ and $PA_4[BR_q(s_3), BR_q(l_2)]$.

In the evaluation, we use the following formula to compute the approximate probability for a given k-subset A:

$$prob(A) \approx \sum_{o_z \in A} \sum_{x=1}^{g-1} p_{o_z}(PA_x) \prod_{o_i \in A \setminus \{o_z\}} 0.5 \cdot (P_{o_i}(PA_x) + P_{o_i}(PA_{x-1})) \prod_{o_j \in O' \setminus A} 0.5 \cdot (1 - P_{o_j}(PA_x) + 1 - P_{o_j}(PA_{x-1}))dr$$

Here, $p_{o_z}(PA_x)$ is the pdf of o_z in this partition; $P_{o_i}(PA_x)$ ($P_{o_i}(PA_{x-1})$) is the upper (lower) bound cdf of the object o_i whose distance to the query location is in partition PA_x. Average values are used to calculate approximations. Similarly, $1 - (P_{o_j}(PA_x)$ $(1 - P_{o_j}(PA_{x-1}))$ is the upper (lower) bound cdf of the object o_j that is farther than partition PA_x.

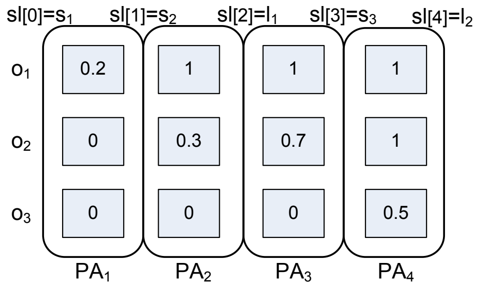
For an object in O', its cdf value in each partition can be computed once and recorded in a two-dimensional array of size $|O'| \times (g-1)$. The cdf value of object o_i in partition PA_x, i.e., $P_{o_i}(PA_x)$, is evaluated as $P_{o_i}(sl[x])$ using Equation 1.

$$P_{o_i}(PA_x) = P_{o_i}(sl[x])$$

If $sl[x] < s_i$, $P_{o_i}(PA_x)$ equals 0 because there is no chance for o_i appearing in the partition. On the other hand, if $sl[x] \geq l_i$, $P_{o_i}(PA_x)$ equals 1 because the object o_i must appear nearer than the partition. The pdf value of object o_i in partition PA_x can be evaluated as the difference between the cdf value in the current partition and the cdf in the previous partition.

$$p_{o_i}(PA_x) = P_{o_i}(PA_x) - P_{o_i}(PA_{x-1}), \quad x > 1$$

For the special case where $x = 1$, $p_{o_i}(PA_1)$=$P_{o_i}(PA_1)$. The cdfs of the running example are shown in Figure 9.



	PA1	PA2	PA3	PA4
o1	0.2	1	1	1
o2	0	0.3	0.7	1
o3	0	0	0	0.5

Figure 9: Partition Based CDF Values

6. EMPIRICAL STUDY

6.1 Experimental Settings

Synthetic Data Set We generate moving objects using a 3-floor building plan with 30 rooms and 3 staircases on each floor. All rooms and staircases are connected by doors to a hallway in a star-like manner. An RFID reader is deployed by the door of each room. In addition, readers are deployed along the hallways and in the staircases. A total of 143 RFID readers are deployed: the readers deployed by doors are undirected partitioning devices; and those deployed along the hallways and in the staircases are presence devices.

Three rules are used to generate movements: 1) an object in a room can move to the hallway or move inside the room; 2) an object in a staircase can move to the hallway or move in the staircase; 3) an object in the hallway can move in the hallway, move to one of the staircases, or move to one of the rooms. At each step, an object randomly chooses a room as the destination. If the destination room chosen is on the same floor as the object, it will move according to MIWD. Otherwise, it will use the nearest staircase. When the object enters the destination room, it will move inside the room for a random time duration and then start a new movement.

Real Data Set Over 1,000,000 tracking records are collected each day from 25 Bluetooth hotspots in Copenhagen Airport. We extract the tracking data on the most active day between April 2008 and October 2008. As a result, over 1.1M tracking observations are recorded in about 110K sampling units for a total of 9,638 moving objects, i.e., individuals with Bluetooth enabled devices.

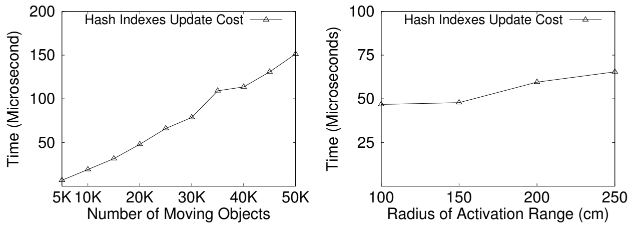
We run all experiments on a Windows XP Pro enabled PC with a 2.66GHz Core2 Duo CPU and 3.25GB main memory.

6.2 Costs of Indoor Moving Object Indexing

We first evaluate the performance of the proposed hashed based indexing structures with respect to the synthetic data.

We implement the object sets in these hash tables as bitmaps, which require less memory space and are update-efficient. We use a 4-byte *int* value for each table key (object, device, cell identifier). A 6,250 byte bitmap is enough for representing the largest 50K objects in our setting. As a result, each entry in DHT, $CDHT$, $CNHT$ is 6,254 bytes. Therefore, 143 RFID readers and 97 cells need (143+97·2)·6254=2.1M bytes memory. The device identifier and timestamps in $OLHT$ are represented as *int* values. Thus, each entry in $OLHT$ needs 12 bytes, and 50K objects need 50K·12 = 600K bytes of memory, a modest memory consumption.

At each sampling unit, the costs of updating these memory resident hash tables are insignificant, as reported in Figure 10. As the number of objects increases, the update cost increases slowly according to Figure 10(a). Note the cost for 50K objects is still very low. Figure 10(b) shows that varying the activation range does not affect the update cost significantly.



(a) Effect of Object Numbers (b) Effect of Activation Range

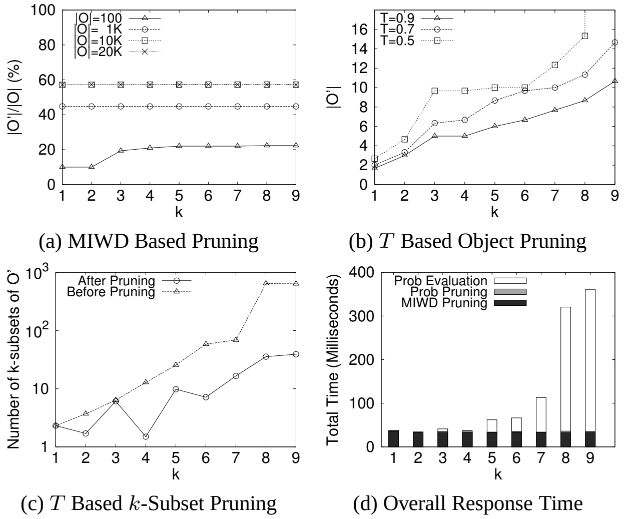
Figure 10: Update Efficiency of Indexes

6.3 Pruning Effectiveness and Query Efficiency

Using the synthetic data set, we fix the device activation range at 100 cm, and 20K objects and threshold $T = 0.9$ are used unless stated otherwise. We choose 20 random indoor positions as kNN query locations. We vary k from 1 to 9. For each query location, 50 kNN queries are issued with different timestamps. We report the average results over all these queries.

To measure the effectiveness of the MIWD-based pruning, we record the ratio of object reduction, i.e., $|O'|/|O|$. The results are reported in Figure 11(a). For $|O| = 100$, only about 20% of the objects are left in the candidate set O' after pruning. For larger $|O|$s, the pruning ratio is still as high as around 50%. This indicates that the distance based pruning is very effective. For larger $|O|$s, the ratio stays constant as k varies because indoor objects overlap much more than do outdoor objects. For example, after some time, the uncertain regions of all objects that left $device_{20}$ are in cell c_{20} (in Figure 8).

We measure the effectiveness of probability threshold based pruning using two metrics. First, we measure $|O'|$ as the pruning is able to eliminate unqualified objects (lines 1–3 in Algorithm 7). According to the results shown in Figure 11(b), for higher threshold T, only very few objects remain in O' after the pruning. Second, we compare the number of qualified k-subsets of O' before and after the pruning. According to the results reported in Figure 11(c), a significant portion of k-subsets is eliminated by the pruning. These results indicate that probability threshold based pruning is very effective.



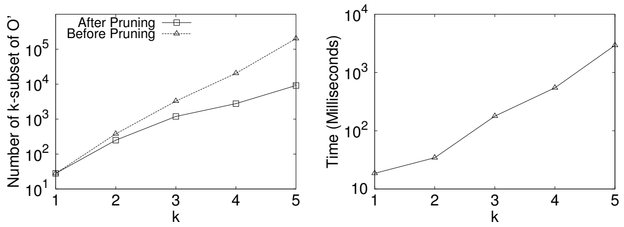
(a) MIWD Based Pruning (b) T Based Object Pruning

(c) T Based k-Subset Pruning (d) Overall Response Time

Figure 11: Pruning Effectiveness and Query Efficiency on Synthetic Data

The results on overall query response time are reported in Figure 11(d). Probability threshold based pruning is efficient because fast estimates are calculated. For larger k values, the time spent on the probability evaluation is much higher than the other two, which is due to: (1) Our partition based estimation (can be regarded as coarse-grained numerical integration) is more time consuming than "Prob Pruning" and "MIWD Pruning". (2) For large k, fewer candidate k-subsets are filtered out in pruning step 2, so more k-subsets need prob evaluation.

We also test our PTkNN query processing techniques on the real data. We choose 5 Bluetooth hotspot locations as query locations and issue kNN query with different k value at 100 separate timestamps. The results on the effectiveness of the probability threshold based pruning are shown in Figure 12(a). Larger k values render the pruning more effective. The results on overall query response time are reported in Figure 12(b). Larger k values result in more k-subsets, which call for more probability evaluations.



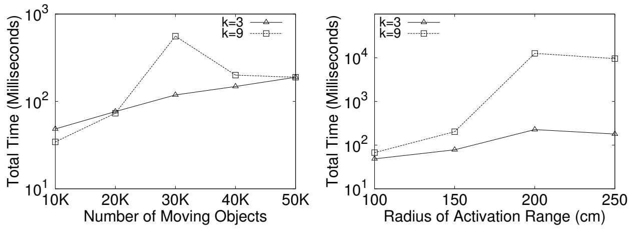
(a) T based k-subset Pruning (b) Overall Response Time

Figure 12: Results on Real Data

6.4 Query Processing Scalability

In this part, we evaluate the scalability of query processing using the synthetic data set. First, we fix the activation range radius at 100 cm, and then vary the object numbers from 10K to 50K. As shown in Figure 13(a), the total query response time increases steadily for $k = 3$. The increase for $k = 9$ at 30K objects is attributed to the high probability evaluation cost (See Figure 11(d)).

Second, we fix the number of objects at 10K and vary the radius of the activation range from 100 cm to 250 cm. The resulting total query response times are reported in Figure 13(b). Larger ranges have two effects: larger imprecise uncertain regions for the moving objects and more active objects being detected by positioning de-



(a) Effect of Object Number (b) Effect of Activation Range

Figure 13: Query Processing Scalability

vices. The former tends to prolong the query processing time due to the uncertain region based probability calculations. The latter tends to have the opposite effect because active objects' uncertain regions become much simpler. Therefore, Figure 13(b) exhibits increases from 100 cm to 200 cm followed by slight decreases at 250 cm.

7. CONCLUSION AND FUTURE WORK

Given an indoor location q and a probability threshold T, a probabilistic threshold kNN (PTkNN) query returns all subsets of k indoor moving objects that have probability larger than T of containing the kNN query result of q. The paper proposes a complete set of techniques for computing PTkNN queries. We propose the minimum indoor walking distance (MIWD) as the distance metric for indoor spaces. Assuming symbolic indoor positioning, we design a hash-based indexing scheme for indoor moving objects. We then formalize the uncertainty of indoor moving object locations. On these foundations, we propose MIWD based pruning, probability threshold based pruning, and efficient probability evaluation for processing PTkNN queries. Finally, we conduct a comprehensive empirical study using both synthetic and real data. The results show that the proposed techniques are effective, efficient, and scalable.

Some interesting research directions exist. As discussed in Section 5.3, analyzing historical trajectory data may discover associations among object movements, which can be used to design more efficient group pruning in processing a PTkNN query. Regarding the uncertainty model of indoor moving objects, it is also interesting to conduct probabilistic analysis on other kinds of object distributions, e.g., Gaussian distribution.

Acknowledgments

This research was partially supported by the Indoor Spatial Awareness project of the Korean Land Spatialization Group and BK21 program. C. S. Jensen is an Adjunct Professor at University of Agder, Norway. His work was done in part when he was a Visiting Scientist at Google Inc.

8. REFERENCES

[1] P. Bahl and V. Padmanabhan. RADAR: an in-building RF-based user location and tracking system. In *Proc. INFOCOM*, pp. 775–784, 2000.

[2] C. Becker and F. Dürr. On location models for ubiquitous computing. *Personal Ubiquitous Computing*, 9(1):20–31, 2005.

[3] G. Beskales, M. A. Soliman, and I. F. Ilyas. Efficient search for the top-k probable nearest neighbors in uncertain databases. *PVLDB*, 1(1):326–339, 2008.

[4] H.-J. Cho and C.-W. Chung. An efficient and scalable approach to cnn queries in a road network. In *Proc. VLDB*, pp. 865–876, 2005.

[5] R. Cheng, D. V. Kalashnikov, and S. Prabhakar. Querying imprecise data in moving object environments. *IEEE Trans. Knowl. Data Eng.*, 16(9):1112–1127, 2004.

[6] R. Cheng, L. Chen, J. Chen, and X. Xie. Evaluating probability threshold k-nearest-neighbor queries over uncertain data. In *Proc. EDBT*, pp. 672–683, 2009.

[7] E. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[8] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *Proc. SIGMOD*, pp. 47–57, 1984.

[9] J. Hightower and G. Borriello. Location systems for ubiquitous computing. *IEEE Computer*, 34(8):57–66, 2001.

[10] G. R. Hjaltason and H. Samet. Distance browsing in spatial databases. *ACM TODS*, 24(2):265–318, 1999.

[11] C. S. Jensen, J. Kolář, T. B. Pedersen, and I. Timko. Nearest neighbor queries in road networks. In *Proc. GIS*, pp. 1–8, 2003.

[12] C. S. Jensen, H. Lu, and B. Yang. Graph model based indoor tracking. In *Proc. MDM*, pp. 122–131, 2009.

[13] M. R. Kolahdouzan and C. Shahabi. Voronoi-based k nearest neighbor search for spatial network databases. In *Proc. VLDB*, pp. 840–851, 2004.

[14] D. Li and D. L. Lee. A lattice-based semantic location model for indoor navigation. In *Proc. MDM*, pp. 17–24, 2008.

[15] D. Pfoser and C. S. Jensen. Capturing the uncertainty of moving-object representations. In *Proc. SSD*, pp. 111–132, 1999.

[16] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao. Query processing in spatial network databases. In *Proc. VLDB*, pp. 802–813, 2003.

[17] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *Proc. SIGMOD*, pp. 71–79, 1995.

[18] K. Raptopoulou, A. Papadopoulos, and Y. Manolopoulos. Fast nearest-neighbor query processing in moving-object databases. *GeoInformatica*, 7(2):113–137, 2003.

[19] S. Šaltenis, C. S. Jensen, S. T. Leutenegger, and M. A. Lopez. Indexing the positions of continuously moving objects. In *Proc. SIGMOD*, pp. 331–342, 2000.

[20] Z. Song and N. Roussopoulos. K-nearest neighbor search for moving query point. In *Proc. SSTD*, pp. 79–96, 2001.

[21] Y. Tao and D. Papadias. Time-parameterized queries in spatio-temporal databases. In *Proc. SIGMOD*, pp. 334–345, 2002.

[22] Y. Tao, D. Papadias, and Q. Shen. Continuous nearest neighbor search. In *Proc. VLDB*, pp. 287–298, 2002.

[23] R. Want. *RFID Explained: A Primer on Radio Frequency Identification Technologies*. Morgan and Claypool, 2006.

[24] B. Yang, H. Lu, and C. S. Jensen. Scalable continuous range monitoring of moving objects in symbolic indoor space. In *Proc. CIKM*, pp. 671–680, 2009.

[25] J. Zhang, D. Papadias, K. Mouratidis, and M. Zhu. Spatial queries in the presence of obstacles. In *Proc. EDBT*, pp. 366–384, 2004.