Scalable Continuous Range Monitoring of 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

Indoor spaces accommodate large populations of individuals. The continuous range monitoring of such objects can be used as a foundation for a wide variety of applications, e.g., space planning, way finding, and security. Indoor space differs from outdoor space in that symbolic locations, e.g., rooms, rather than Euclidean positions or spatial network locations are important. In addition, positioning based on presence sensing devices, rather than, e.g., GPS, is assumed. Such devices report the objects in their activation ranges. We propose an incremental, query-aware continuous range query processing technique for objects moving in this setting. A set of critical devices is determined for each query, and only the observations from those devices are used to continuously maintain the query result. Due to the limitations of the positioning devices, queries contain certain and uncertain results. A maximum-speed constraint on object movement is used to refine the latter results. A comprehensive experimental study with both synthetic and real data suggests that our proposal is efficient and scalable.

Categories and Subject Descriptors

H.2.8 [**DATABASE MANAGEMENT**]: Database Applications

General Terms

Algorithms, Design, Experimentation, Management

Keywords

Indoor Moving Objects, Symbolic Indoor Space, Continuous Range Monitoring

1. INTRODUCTION

People spend large parts of their lives in indoor spaces such as office buildings, shopping centers, conference facilities, airports, and other transport infrastructures. Meanwhile, such spaces are becoming increasingly large and complex. For example, the London Underground has 268 stations and a network of 408 kilometers [1]. Each hour, 146,000 passengers enter its tube system: during the three morning peak hours alone, 51,100 people enter the busiest tube station, Waterloo; and the total number of daily passengers exceeds 4 million.

With the deployment of indoor positioning based on technologies such as RFID [19] and Bluetooth [8], it is possible to continuously monitor indoor moving objects in order to support various applications. For example, such monitoring is very useful for space use analysis and security purposes. However, existing techniques for continuous range monitoring in outdoor spaces [10, 15, 16] are not easily applicable in indoor spaces, for two main reasons.

First, an indoor space is typically modeled differently from an outdoor space, where either an Euclidean space or a spatial network is typically assumed. Indoor space is characterized by entities such as doors, rooms, and hallways that enable and constrain movement. This renders movement more constrained than movement in Euclidean spaces. Consequently, geometric movement representations, e.g., the linear model that is widely adopted for outdoor movement, are not suitable for indoor movement. Further, indoor movement is less constrained than movement in a spatial network, where objects are constrained to a polyline. As a result, symbolic models, rather than geometric models, of indoor space are often used [3].

Second, proximity-based indoor positioning technologies differ fundamentally from those typically assumed in outdoor settings. Unlike GPS and cellular positioning technologies that are capable of continuously reporting the position and velocity of an object with varying accuracies, proximity-based indoor positioning technologies are unable to report velocities or accurate locations [9]. In particular, an indoor object is detected only when it enters the activation range of a position sensing device, e.g., an RFID reader or a Bluetooth hotspot.

To the best of our knowledge, this paper represents the first work on the continuous monitoring of moving objects in symbolic indoor space. The paper's contribution is fourfold. First, it proposes an infrastructure for indoor range monitoring, which includes a state classification of the moving objects and a hashing-based object indexing scheme that exploits the states.

Second, a query-aware scheme is proposed for the incremental maintenance of range queries. For each query, critical devices are determined so that only the new observations from those devices are needed in order to maintain the query's result. The partitioning of critical devices into five classes enables efficient update.

Third, we provide query results with certain results and uncertain results. Probabilities for the uncertain results are derived from assumed maximum object speeds.

Fourth, the paper reports results of a comprehensive performance study of the paper's proposals using both synthetic and real data.

The remainder of the paper is organized as follows. Section 2

CIKM'09, November 2–6, 2009, Hong Kong, China.

covers preliminaries and Section 3 elaborates on the management of indoor moving objects. Section 4 presents the proposals for the processing of range monitoring queries. Section 5 covers the empirical studies. Finally, Section 6 briefly reviews the related work and Section 7 concludes and discusses research directions.

2. PRELIMINARIES

A simplified plan of the first floor of the CS Department at Aalborg University is shown in Figure 1. The floor is divided into three clusters, each having its own (numbered) hallway and rooms. The clusters are connected by a common hallway, labeled 40. Other floors can be reached via a staircase, labeled 50. The outside is labeled 0. For simplicity, we regard hallways and staircases as rooms. For example, we use "room 10" for "hallway 10."

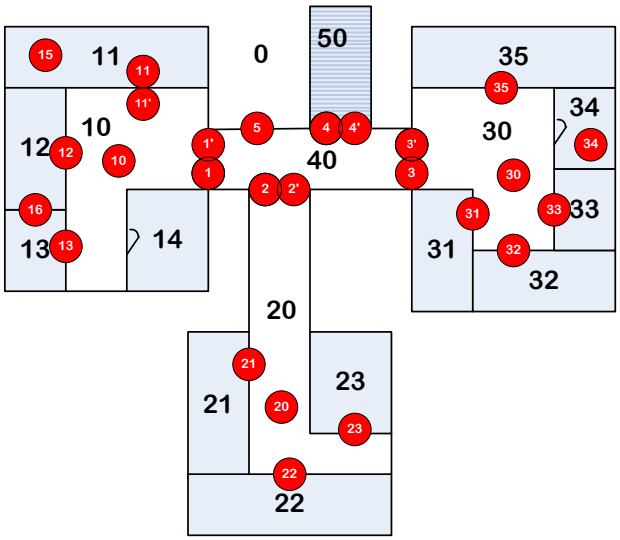


Figure 1: Floor Plan and Positioning Device Deployment

2.1 Symbolic Indoor Positioning

We assume the use of presence, or proximity-based, sensing technologies such as RFID, Bluetooth, and Infrared. We do not consider signal strength [2] as the activation ranges of RFID readers in our setting are relatively small (tens of centimeters to 3 meters [19]).

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 devices and their activation ranges. 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 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 $device_1$. Unlike overlapping devices, so-called paired devices, covered in Section 2.2, are used to detect movement direction, e.g., entry/exit of a room.

For each moving object, only its first and last appearances in the range of a device are of interest. We thus introduce a pre-processing module in-between the sensing devices and the continuous query processing module that continuously (according to the sampling unit T_s) receives readings from all positioning devices, and outputs records in the format $(deviceID,\ objectID,\ t,\ flag)$. Here, $flag = ENTER$ indicates that the object is entering the device's activation range; $flag = LEAVE$ indicates the object is leaving the range. The $deviceID$ can be that of 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.

2.2 Positioning Device Deployment Graph

We differentiate between two types of positioning 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 two 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 devices, which enables the movement direction of an object to be inferred by the reading sequence. An example is $device_{11}$ and $device_{11'}$ in Figure 1.

Next, *presence devices* (*PR*) simply sense the presence of objects in their ranges. These are exemplified by $device_{10}$ in Figure 1.

A *Devices* mapping maintains the information on the positioning devices:

$$Devices:\ \Sigma_{devices} \rightarrow \{(ActRange,\ TYPE)\},$$

where $\Sigma_{devices}$ is the domain of device identifiers, $ActRange$ indicates the activation range of a device (a geometry describing the range); and $TYPE$ indicates the type of a device: *UP*, *DP*, or *PR*.

To facilitate object tracking and querying, a deployment graph $G = (C, E, \Sigma_{devices}, \ell_E)$ is created based on the topological relationship between the floor plan and the positioning device deployment. The set of vertices C consists of the cells formed by the partitioning devices. The set of edges E consists of sets $\{c_i, c_j\}$, where $c_i, c_j \in C$. Further, ℓ_E: $E \rightarrow 2^{\Sigma_{devices}}$ maps an edge to a set of positioning devices: a non-loop edge $\{c_i, c_j\}$ is mapped to the device(s) that partitions cells c_i and c_j, and a loop edge $\{c_i, c_i\}$ is mapped to the presence device(s) in cell c_i.

An in-depth study on deployment graphs, including relevant data structures and algorithms is available elsewhere work [7]. The deployment graph of Figure 1 is shown in Figure 2, where the label D_i indicates the positioning device $device_i$.

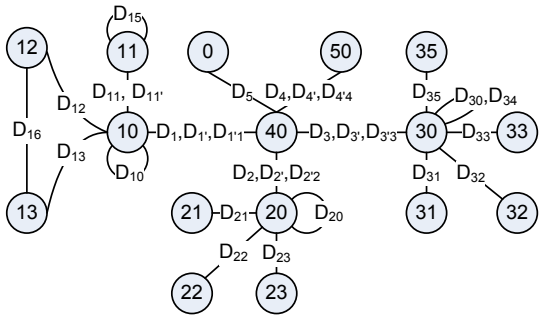


Figure 2: Positioning Device Deployment Graph

Each cell created by a reader deployment corresponds to one or more rooms. For example, cell c_{10} is mapped to rooms 10 and 14 because an object can go between these rooms without being observed. A mapping $Cells: C \rightarrow 2^{\Sigma_{rooms}}$ maintains this relationship, where Σ_{rooms} is the domain of room identifiers.

It is important to observe that rooms make up a partitioning of a floor plan that is 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 any intersecting devices. In the 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.

3. MANAGEMENT OF INDOOR OBJECTS

3.1 States of Indoor Moving Objects

A deployment of positioning devices induces an *active subspace* that is the union of the activation ranges of all the positioning devices and an *inactive subspace* that is the part of space that is not covered by any positioning device. An object is said to be in the *active (inactive) state* when it is in the active (inactive) subspace.

Using the $Devices$ mapping from Section 2.2, we are able to directly determine the whereabouts of active objects. For inactive objects, additional processing and information are needed to infer their possible locations.

We refine the inactive state so that an object is in the *deterministic state* if it is certain that the object is in one specific cell; it is in the *nondeterministic state* if it can be in several cells.

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

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

If an object enters the range of a positioning device, its state changes from inactive (deterministic or nondeterministic) to active.

An object cannot switch directly between deterministic and nondeterministic. For an object to enter/leave a cell, it must be detected by a partitioning device, which makes its state become active before it can switch.

Based on the resulting state diagram, in Figure 3, we proceed to present specific object indexing structures.

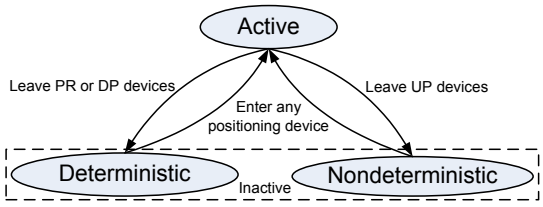


Figure 3: Indoor Moving Object State Diagram

[1] $G.\ell_E^{-1}$ is the reverse function of $G.\ell_E$ introduced in Section 2.2. For simplicity, we throughout the paper 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.

3.2 Indexing Indoor Moving Objects

The partitioning devices render the indoor space discrete. This and the specifics of the positioning devices render indexes for free-moving outdoor objects unsuitable for indoor objects.

We propose an indexing scheme that utilizes several hash tables. Let O_{indoor} be the set of all the moving objects in the indoor space of interest. A *Device Hash Table (DHT)* is created that maps each positioning device, identified by $deviceID$, to the set of active objects in its range:

$$DHT[deviceID] = O_A; \quad deviceID \in \Sigma_{devices}, O_A \subseteq O_{indoor}.$$

Next, a *Cell Deterministic Hash Table (CDHT)* maps each cell, identified by $cellID$, to the set of deterministic objects in it:

$$CDHT[cellID] = O_D; \quad cellID \in C, O_D \subseteq O_{indoor}.$$

Similarly, a *Cell Nondeterministic Hash Table (CNHT)* maps a cell to the set of nondeterministic objects in it:

$$CNHT[cellID] = O_N; \quad cellID \in C, O_N \subseteq O_{indoor}.$$

Finally, an *Object Hash Table (OHT)* captures the states of all objects:

$$OHT[objectID] = (STATE,\ t,\ IDSet);\ objectID \in O_{indoor}.$$

Here $STATE$ denotes the object's current state; t is the start time 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 consisting of a device identifier. If the state is deterministic, $IDSet$ is a singleton set consisting of a cell identifier. If the state is nondeterministic, $IDSet$ is a set of cell identifiers.

The four hash tables need updating whenever there is a new output from the pre-processing module. The update algorithm, described in Algorithm 1, handles a record received from the pre-processing module according to its $flag$ value.

Algorithm 1 updateHashTables(Pre-processing output O, DeploymentGraph G)

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

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 the object 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 a UP device, its state becomes 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 becomes deterministic, and the object is added into the corresponding cell's $CDHT$ (lines 22–25).

The implementation uses bitmaps for storing the sets of object identifiers in DHT, $CDHT$, and $CNHT$. In particular, each value in these hash tables maintains a bitmap, each bit of which corresponds to a specific object. A bit is set only if the object is currently in the object set of the value. Bitmaps require little space, rendering main-memory storage of these hash tables possible. Combined with suitable masks, bitwise AND and OR operations render updates (the insertions and deletions in Algorithm 1) very efficient.

4. CONTINUOUS RANGE MONITORING

4.1 Query Definition and Solution Overview

A *Continuous Range Monitoring Query* ($CRMQ$) takes an indoor spatial range R as parameter. It is activated when it is registered in the system, say t_s. At each point in time, it then reports all objects that are currently within R. This continues until the query is unregistered from the system, say t_e. The query result is maintained from time t_s to time t_e as follows.

$$\forall t \in [t_s, t_e]\text{: } o \in CRMQ[R](\mathcal{M}) \Leftrightarrow o \in \mathcal{M} \wedge pos_{\mathcal{M}}(o,t) \in R,$$

where $\mathcal{M}$ indicates all the objects moving in the indoor space; $pos_{\mathcal{M}}$ is a function which can determine the location of the object o at timestamp t. Multiple monitoring queries may be expected to coexist in the system. They can be registered (unregistered) at different times.

The result of a $CRMQ$ needs updating whenever an object enters or leaves its range. A naive approach is to reevaluate each query when a new observation is produced by the pre-processing module. However, this yields an unnecessarily high workload, especially when the number of concurrent queries is high. Another solution entails periodical reevaluation, which searches the index structures and computes an up-to-date result periodically, according to user configurations. These two approaches are query blind, as they do not take advantage of the ranges of the registered queries when they process the queries.

We propose a query-aware and incremental approach. The idea is that not every pre-processing observation causes changes to the result of every $CRMQ$. For each query, we identify the *critical devices* from which new observations may change the query's result. Accordingly, only $ENTER$ and $LEAVE$ observations from such devices are needed to correctly update the results of the relevant monitoring queries.

Our approach is shown in Figure 4. When a new query is registered in the system, the hash indexes are searched for all moving objects currently in the query range. The set of critical devices and the result of the query are stored in main memory. As time elapses, new observations will be emitted by the critical devices, and the query result will be updated accordingly in the query processing module, which is detailed in Section 4.3.

The range of a $CRMQ$ can be represented either symbolically or geometrically. It is straightforward to use the device and cell identifiers in the symbolic representation. Through the hash tables DHT, $CDHT$ and $CNHT$, the query result can be obtained directly. This arrangement is tightly integrated with the positioning device deployment, which may not be available to query issuers.

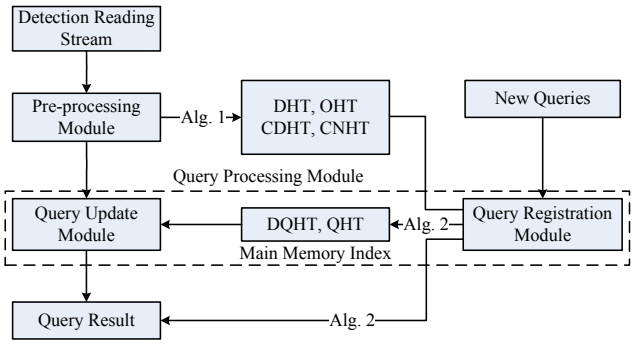


Figure 4: Continuous Monitoring Query Processing

In the geometrical representation, the query range is represented as a geometrical shape, e.g., a polygon or a circle. To enable such queries, the floor plan, positioning device activation ranges, and cells are indexed by spatial indexes, such as a 2-dimensional R-tree, which enables easy retrieval of the corresponding symbolic identifiers. We thus assume that a geometrical query range is not fully contained in any single activation range.

4.2 Query Result Accuracy

In our setting, the position of an object is typically constrained to be in one or several cells, as discussed in Section 3.1. Consequently, the result of a $CRMQ$ is divided into a *certain* and an *uncertain* part. The certain result contains all objects that are definitely in the query range $CRMQ.R$, and the uncertain result contains those objects that may be in the query range.

Specifically, if the query range $CRMQ.R$ covers (intersects) the whole activation range of a device, all active objects in the device's DHT are in the certain (uncertain) result the of $CRMQ$. For example, the range of a $CRMQ$ $query_1$ is shown as a dashed rectangle in Figure 5. The active objects in $device_{13}$ ($device_{16}$) are in the certain (uncertain) result.

We say that region x *covers* region y if y is fully contained in x. We say that x *intersects* with y if they overlap but do not cover each other. Let $\sqcap$ return the intersection of two argument regions. If $x \sqcap y \neq \emptyset$ and neither $x \sqcap y = x$ nor $x \sqcap y = y$, we say that x intersects with y; if $x \sqcap y \neq \emptyset$ and $x \sqcap y = y$, we say that x covers y.

Furthermore, if the query range covers an entire cell, the deterministic (nondeterministic) objects in the cell's $CDHT$ ($CNHT$) are in the certain (uncertain) result. Refer again to the $query_1$ in Figure 5. The deterministic (nondeterministic) objects in c_{13} are in the certain (uncertain) result. If the query range intersects a cell, all objects in the cell, deterministic or nondeterministic, are in the uncertain result of the query. The deterministic and nondeterministic objects in c_{12} are in the uncertain result of $query_1$.

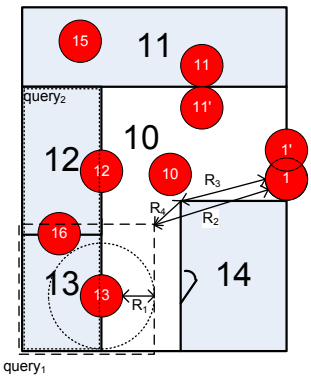


Figure 5: Query Examples

The situation is more complicated when the query range covers more than one cell. Some nondeterministic objects involved can also be in the certain result. Consider $query_2$ whose range is rooms 12 and 13. The query range covers both c_{12} and c_{13}. If an object leaves $device_{16}$, it becomes a nondeterministic object for both c_{12} and c_{13}. However, it is still definitely within the query range and thus in the certain result. For a nondeterministic object, if all its possible cells (can be obtained from OHT) are covered by

the query range, it is included in the certain result of the query.

We proceed to elaborate on the query processing, covering the computation of the initial certain and uncertain results and their incremental maintenance. A probabilistic accuracy analysis on the uncertain result is given in Section 4.4.

4.3 Query Processing

When a new query arrives, the *query registration module* computes the initial query result (both certain and uncertain), identifies the critical devices, and registers the query into the system with the corresponding information. The *query update module* is responsible for incrementally maintaining query results as new observations are emitted by the pre-processing module. See Figure 4.

In Section 4.3.1, we identify for each query critical devices that enable incremental query result update. In Section 4.3.2 we determine the initial result for a new coming query. In Section 4.3.3, we propose the incremental query result update method. In Section 4.3.4, we discuss how to improve query result when the maximum object speed is available.

4.3.1 Critical Devices

For a $CRMQ$ query, a critical device is one from which a new observation can potentially change the query result (either certain or uncertain). In order to continuously update the query result, it is crucial to know all such devices. This is achieved by means of the deployment graph covered in Section 2.2. Clearly, the devices whose activation ranges intersect with or are covered by the query range $CRMQ.R$ are critical.

A query range may also intersect with or cover a set of cells C_{ic} =$\{c|c \sqcap R \neq \emptyset\}$. In order to guarantee the accuracy of the query result, we introduce another extended set of cells C_{ex}=$\{c|\{c, c'\} \in G.E, c' \in C_{ic}\}$, which contains the neighbor cells of C_{ic}. For example, for $query_2$ in Figure 5, C_{ic} = $\{c_{12}, c_{13}\}$; and C_{ex} = $\{c_{10}\}$. Taking the deployment graph into account, all edges whose vertices contain one of the cells in set C_{ic} or C_{ex} indicate the positioning devices whose future observations can potentially change the query result. Critical devices cannot come from any further-away graph edges because for any object to enter or leave the query range, it must first be detected by devices whose corresponding graph edges are closer to the query range.

Let R be the range of a $CRMQ$, d be a positioning device, and d' be the activation range of the device d. In other words, we have d'=*Devices*$(d).ActRange$. To improve the query processing, we categorize all critical devices into five classes.

$$
\begin{aligned}
CLASS1 &= \{d|d' \sqcap R = d'; \forall c \in G.\ell_E^{-1}(d)(c \sqcap R = c)\}\\
CLASS2 &= \{d|d' \sqcap R = d'; \exists c \in G.\ell_E^{-1}(d)(c \sqcap R \neq c;\ c \sqcap R \neq \emptyset)\}\\
CLASS3 &= \{d|d' \sqcap R \neq d';\ d' \sqcap R \neq \emptyset\}\\
CLASS4 &= \{d|d' \sqcap R = \emptyset; \exists c \in G.\ell_E^{-1}(d)(c \in C_{ic})\}\\
CLASS5 &= \{d|\ d' \sqcap R = \emptyset; \forall c \in G.\ell_E^{-1}(d)(c \notin C_{ic});\\
&\qquad \exists c \in G.\ell_E^{-1}(d)(c \in C_{ex})\}
\end{aligned}
$$

The classes have the following pertinent properties. The activation range of a *CLASS1 device* is fully covered by the query range R and all its corresponding cells are fully covered by the query range R. For $query_2$ in Figure 5, $device_{16}$ is a $CLASS1$ device.

The activation range of a *CLASS2 device* is fully covered by the query range R, and at least one of its corresponding cells is not fully covered by the query range R. For example, $device_{13}$ is a $CLASS2$ devices of $query_1$.

The activation range of a *CLASS3 device* intersects with the query range R. For example, $device_{16}$ is a $CLASS3$ devices of $query_1$.

The activation range of a *CLASS4 device* is disjoint with the query range R and at least one of its corresponding cells is in C_{ic}. For example, $device_1$ is a $CLASS4$ device of $query_1$.

Finally, the activation range of a *CLASS5 critical device* is disjoint with the query range R and at least one of its corresponding cells is in C_{ex}, but none of them are in C_{ic}. For example, $device_{10}$ is a $CLASS5$ critical device of $query_2$.

As discussed, only new observations from the critical devices can affect the result of a given query. The classification will be helpful for the query result update (to be detailed in Section 4.3.3). It is then beneficial if the relationships between a query and its critical devices are recorded appropriately in the system. Therefore, a *Device Query Hash Table (DQHT)* is defined:

$$DQHT[deviceID] = \{(queryID, CLASS)\},$$

where $deviceID$ indicates a device, $queryID$ indicates a query that has the device as a critical device, and $CLASS$ indicates the class of the critical device in the particular query.

During continuous query updating, as soon as a new observation from a device is output by the pre-processing module, all queries relevant to the device are determined through *DQHT*. This can avoid unnecessary result updates on queries for which the device is not critical.

4.3.2 Query Registration

In order to process multiple concurrent CRMQ efficiently, an incremental query result update method is applied. In other words, each update is based on the previous query result. A query index *Query Hash Table(QHT)* is created in main memory. It maps a query identifier to the query's results:

$QHT[queryID] = (CR, UR)$; $CR \subseteq O_{indoor}$, $UR \subseteq O_{indoor}$,

where CR is the certain result and UR is the uncertain result.

The registration procedure for a new query, detailed in Algorithm 2, identifies the critical devices, obtains the initial query results, and registers the relevant information to facilitate future result updates. First, a new identifier is generated for the query (line 5). The covered devices/cell sets and intersected devices/cell sets are determined through the predefined spatial index (lines 6–9). The covered devices are then added to the critical devices set, and the class is determined according to the relevant definitions (lines 10–14). The intersected devices are added into the critical devices set with $CLASS3$ (lines 15–16). After that, $CLASS4$ critical devices are determined (lines 17–20). For each edge in the deployment graph G, if one of its two vertices is in the covered or intersected cell set and the edge's corresponding device is not in critical devices set, the device is a $CLASS4$ critical device. At the same time, the extended cell set C_{ex} is determined (line 21). For each edge in G, if one of its two vertices is in the extent cell set C_{ex}, and the edge's corresponding device is not in critical devices set, the device is a $CLASS5$ critical device (lines 22–25).

For each device in the covered device set, all the corresponding active objects from DHT are added to the certain result (lines 26–27). For each device in the intersected device set, all the corresponding active objects from DHT are added to the uncertain result (lines 28–29). For each cell in the covered cell set C_c, all the corresponding deterministic objects from $CDHT$ are added to the certain result (lines 30–31). If the covered cell set has more than one cells, each nondeterministic object in these cells is checked. If all its possible cells $OHT[o].IDSet$ are in C_c, the object is added to the certain result. Otherwise, the object is added to the uncertain result (lines 32–37). If C_c has only one cell, all the corresponding nondeterministic objects from $CNHT$ are added to the uncertain result (lines 38–39). For each cell in the intersected cell set, both deterministic objects from $CDHT$ and nondeterministic objects from $CNHT$ are added to the uncertain result (lines 40–41). Then, the initial result sets are added to QHT (line 42). Finally,

the $DQHT$ is updated for each critical device (line 43-44).

Algorithm 2 register (Range R, DeploymentGraph G)

```
1: deviceSet D_c←∅, D_uc←∅;
2: cellSet C_c←∅, C_uc←∅, C_ex ← ∅;
3: objectSet R_c←∅, R_uc←∅;
4: CriticalDeviceList(deviceID, CLASS) cd←∅;
5: Generate a new identifier queryID for the query;
6: D_c ← Devices that are covered by R;
7: D_uc ← Devices that intersect with R;
8: C_c ← Cells which are covered by R;
9: C_uc ← Cells that intersect with R;
10: for each device d in D_c do
11:     if all the cells in G.ℓ_E^{-1}(d) are in C_c then
12:         Add (d, CLASS1) to cd;
13:     else if one of the cells in G.ℓ_E^{-1}(d) is in C_uc then
14:         Add (d, CLASS2) to cd;
15: for each device d in D_uc do
16:     Add (d, CLASS3) to cd;
17: for each edge e in G do
18:     if (C_c ∪ C_uc) ∩ e ≠ ∅ AND (C_c ∪ C_uc) ∩ e ≠ (C_c ∪ C_uc) then
19:         if G.ℓ_E(e) ∉ cd.deviceID then
20:             Add (G.ℓ_E(e), CLASS4) to cd;
21:         C_ex ← C_ex ∪ e \ (C_c ∪ C_uc);
22: for each edge e in G do
23:     if C_ex ∩ e ≠ ∅ then
24:         if G.ℓ_E(e) ∉ cd.deviceID then
25:             Add (G.ℓ_E(e), CLASS5) to cd;
26: for each device d in D_c do
27:     R_c ← R_c ∪ DHT[d];
28: for each device d in D_uc do
29:     R_uc ← R_uc ∪ DHT[d];
30: for each cell c in C_c do
31:     R_c ← R_c ∪ CDHT[c];
32: if |C_c| > 1 then
33:     for each nondeterministic object o in C_c do
34:         if OHT[o].IDSet ⊂ C_c then
35:             Add o into R_c;
36:         else
37:             Add o into R_uc
38: else
39:     R_uc ← R_uc ∪ CNHT[c];
40: for each cell c in C_uc do
41:     R_c ← R_c ∪ CDHT[c];  R_uc ← R_uc ∪ CNHT[c];
42: QHT[queryID] ← (R_c, R_uc);
43: for each item a in cd do
44:     Add (queryID, a.CLASS) into DQHT[a.deviceID];
```

4.3.3 Query Result Updates

When the pre-processing module outputs a new observation ($deviceID$, $objectID$, t, $flag$), the result of each query having $deviceID$ as a critical device needs updating. The query update cases are summarized in Table 1, where CR (UR) is the certain (uncertain) result and the query result after each update is described. For simplicity, we use o to denote the moving object identified by $objectID$.

Upon receipt of a new observation O, the query update module only updates the results of those queries that $O.deviceID$ maps to in $DQHT$. For each such query, action is taken according to Table 1.

If O is an $ENTER$ observation, the observed object is entering the activation range of the corresponding critical device. For a $CLASS1$ or $CLASS2$ critical device, whose range is covered by the query range, the object is definitely in the query range, so the object is added to the certain result of the query. If the object is originally in the uncertain result set, it is deleted. For a $CLASS3$ critical device, whose range intersects with the query range, the object is possibly in the query range. Therefore, the object is added to the uncertain result of the query. If the object is already in the certain result set, it is deleted. For a $CLASS4$ or $CLASS5$ critical device, whose range is disjoint from the query range, the object is definitely not in the query range. Therefore, the object is deleted from the certain or uncertain result of the query as necessary.

Table 1: Query Updates w.r.t. Critical Devices

	ENTER	LEAVE
CLASS1	$CR \cup \{o\}$, $UR \setminus \{o\}$	CR, UR
CLASS2	$CR \cup \{o\}$, $UR \setminus \{o\}$	$CR \setminus \{o\}$, $UR \cup \{o\}$
CLASS3	$CR \setminus \{o\}$, $UR \cup \{o\}$	CR, UR
CLASS4	$CR \setminus \{o\}$, $UR \setminus \{o\}$	CR, $UR \cup \{o\}$
CLASS5	$CR \setminus \{o\}$, $UR \setminus \{o\}$	CR, UR

If O is a $LEAVE$ observation, the observed object is leaving the activation range of the corresponding critical device. After leaving a $CLASS1$ critical device's range, all the possible cells in which the object can be are fully covered by the query range, so the object is still definitely in the query range. Therefore, the object should remain in the certain result, which means that the query result needs not be updated. After leaving a $CLASS2$ critical device's range, the object may enter the adjacent cell that is not fully covered by the query range. Thus the object may not be in the query range. Therefore the object is added to the uncertain result and removed from the certain result if necessary.

After leaving a $CLASS3$ critical device's range, cells in which the object may be intersect with the query range. The object should be still in the uncertain result set. Therefore, the query result needs no update. After leaving a $CLASS4$ critical device's range, cells in which the object may be intersect with the query range. The object is therefore added to the uncertain result set. After leaving a $CLASS5$ critical device's range, the object must be still out of the query range. Therefore, the query result needs no update.

4.3.4 Deferred Query Result Updates

If the maximum speed of a moving object is known, the query result update can be deferred, which improves query accuracy.

Consider $query_1$ in Figure 5. According to Table 1, after an object o leaves a $CLASS2$ critical device $device_{13}$, o should be moved from the certain to the uncertain result. Let object o's maximum speed be V_{max}, and let the time span from its latest $LEAVE$ observation on $device_{13}$ to current time be Δt. The longest possible distance o can move from the boundary of the $device_{13}$'s activation range is R_1=$V_{max} \cdot \Delta t$. In other words, the possible region of o is constrained by a circle with the deployed location of $device_{13}$ as the center, and the radius of $device_{13}$ plus R_1 as the radius. If this maximum speed constraint circle is still in the query range, the moving object is still definitely in the query range and the certain result. Consequently, we can maintain the certain result without updating for an extra period of time ΔT.

Assuming that function $minDist(deviceID, R)$ returns the *minimum indoor walking distance* from the boundary of a device's activation range to that of a given query range R, the aforementioned ΔT is determined as $minDist(deviceID, R)/V_{max}$. After time period ΔT, the moving object may leave the range, and we cannot guarantee that it remains in the certain result. Notice that the minimum indoor walking distance is different from the shortest Euclidean distance, as we shall see later.

According to Table 1, if a moving object o leaves a $CLASS4$ critical device, e.g., $device_1$ for $query_1$ in Figure 5, o is added to the uncertain result of the query. As a matter of fact, only after some time ΔT=$minDist(device_1, R)/V_{max}$, can the object o possibly enter the query range. As shown in Figure 5, the function $minDist(device_1, range)$ returns the minimum indoor walking distance, which is $R_3 + R_4$ rather than the Euclidean distance

R_2. Because the Euclidean distance line segment R_2 is intersected by room 14, it is impossible for an object to walk along R_2. Consequently, the uncertain query result can be maintained without updating for a longer time.

However, deferred query result updates based on the maximum speed constraint are only applied to $LEAVE$ observations that are from $CLASS2$ and $CLASS4$ critical devices. Let t be the time when the latest $LEAVE$ observation of o on $deviceID$ is produced and let $\Delta T = minDist(deviceID, R)/V_{max}$.

If $deviceID$ is a $CLASS2$ critical device, object o is kept in the certain result from time t until time $t + \Delta T$ when o will be moved to the uncertain result. We call such an update a $C2U$ deferred update. If $deviceID$ is a $CLASS4$ critical device, object o will not be added to the uncertain result until time $t + \Delta T$. We call such an update an $N2U$ deferred update.

In order to execute deferred updates efficiently, each query maintains a *deferred update table (DUT)* which is a hash table defined as follows:

$$DUT[queryID] = \{(T, objectID,\ TYPE)\},$$

where T indicates the time when the deferred update should be executed, $objectID$ indicates the object involved, and $TYPE$ indicates the type of the deferred update.

The elements in DUT are sorted non-decreasingly on the future update time T. Each time when the query is to be updated, the query processing module needs only to check the first record of DUT to determine whether there is any deferred update to execute. If the object in DUT enters the activation range of any critical device before the deferred update time, the corresponding record should be deleted from DUT.

To incorporate deferred query result updates, the query update module needs slight changes as follows. For an $ENTER$ observation, if the object is in DUT, the corresponding record should be removed from DUT. For a $LEAVE$ observation, if the critical device is a $CLASS2$ device, a $C2U$ update record is created and inserted into DUT with the corresponding future update time. If the device is a $CLASS4$ device, a $N2U$ update record is created and inserted into DUT.

4.4 Probabilistic Analysis of Uncertain Results

Given a range monitoring query and a moving object o in its uncertain result, we intend to infer the probability that o is in the query range R. We assume that the possible locations of an object o in a given indoor space conform a uniform distribution within all reachable regions constrained by o's maximum speed. The binary relationship $o\Theta R$ denotes that the object o is in the range R.

4.4.1 Probabilities for Active Objects

Recall from Section 4.3.2 that all the objects in the activation range of a $CLASS3$ device d form the uncertain result of the relevant query. We first infer the probabilities for such objects based on the areas of the regions in which objects can be. Formally, the probability that an active object o is in the range R is defined as:

$$prob(o\Theta R) = \frac{Area(Devices(d).ActRange \sqcap R)}{Area(Devices(d).ActRange)}.$$

In Figure 5, $device_{16}$ is a $CLASS3$ device for $query_1$, and the probability for an active object in $device_{16}$ to be in the query range is calculated as $\frac{Area(Devices(device_{16}).ActRange \sqcap R)}{Area(Devices(device_{16}).ActRange)}$.

4.4.2 Probabilities for Inactive Objects

As described in Section 4.3.3, after leaving a $CLASS2$, $CLASS3$, or $CLASS4$ critical device, an inactive object is added to or kept in the uncertain result of the query. For such inactive objects, which are currently in cells, the probabilities can be defined based on the maximum speed constraint.

We use the binary relationship $o\Upsilon d$ to indicate that the inactive object o has most recently left critical device d. Let the set $CD234$ contain all the $CLASS2$, $CLASS3$, and $CLASS4$ critical devices for a given query. The probability that an inactive object o is in query range R is defined as:

$$prob(o\Theta R) = \sum_{d_i \in CD234} prob(o\Theta R \mid o\Upsilon d_i) \cdot prob(o\Upsilon d_i)$$

For each object o in the uncertain set, the positioning device d which o has just left can be exactly determined through the query update module. The probability $prob(o\Upsilon d)$ equals 1 for this particular device d and equals 0 for all other devices in $CD234$. Let $\mathcal{H}$ be the assumption that indicates this fact. We then need to consider the probability $prob(o\Theta R, \mathcal{H})$, which indicates the probability that object o is in the query range R given assumption $\mathcal{H}$.

After leaving the positioning device d, the object should enter one of the cells in the set C_d=$G.\ell_E^{-1}(d)$. Therefore, the probability can be defined as follows:

$$prob(o\Theta R, \mathcal{H}) = \sum_{c_i \in C_d} prob(o\Theta R \mid o\Theta c_i, \mathcal{H}) \cdot prob(o\Theta c_i, \mathcal{H})$$

If the device d is a DP or PR device, the object must enter exactly one specific cell. The probability of the object being in this cell is 1. If d is a UP device, the object may enter an arbitrary cell in the cell set C_d. Assume that the function $Bound(deviceID, cellID)$ returns the length of the boundary of the device's activation range that falls in the cell. The probability that an object enters a cell $c_i \in C_d$ is defined as:

$$prob(o\Theta c_i, \mathcal{H}) = \begin{cases} \dfrac{Bound(d, c_i)}{Bound(d)} & \text{if } d \text{ is UP} \\ 1 & \text{otherwise} \end{cases}$$

Let the maximum speed of object o be V_{max} and assume that o left critical device d at time t. Function $Circle(time, deviceID, objectID)$ returns the maximum speed constraint circle with respect to the parameters given. Binary relationship Λ indicates an *indoor accessible intersection*, which returns the accessible intersection of two indoor regions. For simplicity, we use $Circle$ for $Circle(t, d, o)$. We then have:

$$prob(o\Theta R | o\Theta c_i, \mathcal{H}) = \begin{cases} \dfrac{Area(c_i \Lambda Circle \sqcap R)}{Area(c_i \Lambda Circle)}, & c_i \sqcap R \neq \emptyset \\ 0, & c_i \sqcap R = \emptyset \end{cases}$$

Refer to the example in Figure 6. Assume that the assumption $\mathcal{H}$ indicates that object o most recently left $device_{12}$ at time t. The object o is constrained in its reachable region, i.e., the maximum speed constraint circle, which is shown as the dashed circle R_1 to the left in Figure 6. Let the maximum speed be V_{max} and let the time span from the current time to t be ΔT. Then R_1=$\Delta T \cdot V_{max}$. Although the circle with radius R_1 intersects with room 14, the circle does not intersect with the door of room 14. This means that the object cannot enter room 14 within time span ΔT.

Next, let the minimum indoor walking distance from the door to the boundary of $device_{12}$ be l=$minDist(device_{12}, door)$. If $\Delta T < l/V_{max}$, the object cannot enter room 14. Here, $c_{10} \Lambda Circle$ denotes the accessible intersection between the maximum speed constraint circle R_1 and the part of cell c_{10} within room 10. Continuing, $c_{10} \Lambda Circle \sqcap R$ is the intersection between the accessible region $c_{10} \Lambda Circle$ and query range R, shown as the shaded region to the left in Figure 6.

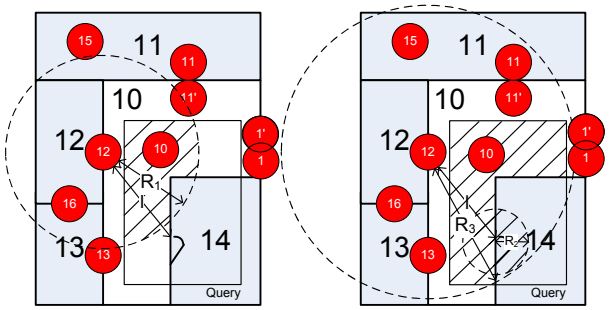


Figure 6: Probabilistic Analysis of Uncertain Results

As time passes, if $\Delta T \geq l/V_{max}$, the moving object can possibly enter room 14. The reachable region in room 14 is the maximum speed constraint circle with R_2=$V_{max} \cdot (\Delta T - l/V_{max})$ as radius and the location of the door as center.

The region of $c_{10}\Lambda Circle$ is composed of two parts. The first part is the intersection of the room 10 part of cell c_{10} and the maximum speed constraint circle R_3=$\Delta T \cdot V_{max}$. The second part is the intersection between the room 14 part of cell c_{10} and the circle with radius R_2. Similarly, $c_{10}\Lambda Circle \sqcap R$ is the intersection region between $c_{10}\Lambda Circle$ and query range R, shown as the shaded region to the right in Figure 6.

Next, we define the relationship Λ. Given a cell c, we can obtain all its rooms through the *Cells* mapping defined in Section 2.2. For a cell with more than one room (indicated by $|Cells(c)| > 1$), we partition it into a *direct part* and an *indirect part*. The direct part is the region reachable without any constraint, e.g., room 10 in cell c_{10} in Figure 6. The indirect part is the region only reachable through some constraint, e.g., room 14 in cell c_{10} is only reachable through the door.

Given a cell c and a device d, function $DP(c, d)$ returns the direct part of c for d, and $IP(c, d)$ returns the indirect part. Note that the same room in the same cell can be a direct or an indirect part depending on the device assumed. Consider cell c_{30} in Figure 1: for $device_{30}$, room 30 is the direct part and room 34 is the indirect part. The arrangement for $device_{34}$ is the opposite.

After leaving a device at time t, the reachable region for object c in a cell c is defined as follows:

$$c\Lambda Circle = (DP(c, d) \sqcap Circle(t)) \bigcup \cup_{m \in IP(c,d)} (m \sqcap Circle(t'))$$

The reachable region involves two parts. The first part is the intersection between the direct part of the cell and the maximum speed constraint circle with time parameter t (that can be obtained from the assumption $\mathcal{H}$).

The second part is the intersection between the indirect part of the cell and the maximum speed constraint circle with time parameter t', the time when the object satisfies the constraint to enter the indirect part. In the running example, t' is the time when object o reaches the door of room 14. This t' is determined by the minimum walking distance l and the maximum speed, i.e., t'=l/V_{max}.

Note that the minimum walking distances from different devices to the same indirect part are different. For example, in Figure 6, the distances from devices $device_{10}$ and $device_{12}$ to the door of room 14 are different. As a final remark, the minimum walking distance from a device to a given indirect part is determined by the floor plan and the positioning device deployment. Such distances can be calculated by the query registration module and recorded in the system for future use.

5. EXPERIMENTAL STUDY

We use both synthetic and real data in the experimental study. 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. According to the definition of partitioning and presence devices in Section 2.2, 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 the minimum indoor walking distance. Otherwise, it will use the nearest staircase. After the object gets into the destination room, it will move inside the room for a random time duration and then start a new movement. All objects move with the constant speed of 4 km/hour. We vary the number of indoor moving objects and the radius of the activation ranges of the positioning devices according to Table 2, with default values shown in bold.

Table 2: Parameter Settings

Number of Objects	1K, **10K**, 20K, 30K, 40K, 50K
Activation Range	**100**, 150, 200, 250 (cm)
Number of Queries	500, **1K**, 2K, 3K, 4K, 5K

We use a real data set collected from Copenhagen Airport. More than 1,000,000 tracking records from 25 Bluetooth hotspots are collected each day. We extract the tracking data on the most active day from April 2008 to October 2008. The total number of moving objects, i.e., those passengers with Bluetooth enabled devices in Copenhagen Airport, is 9,638. More than 1.1M tracking observations are recorded in around 110K sampling units.

In the experimental study, we compare three methods: (1) Naive Method (NM): When a new observation is produced by the preprocessing module, every query is reevaluated. We use this naive method as a baseline. (2) Periodical Method (PM): The query processing module searches the index and updates the query results periodically. The query result is obtained from the corresponding hash tables in the same way as in query registration (Algorithm 2). In the experiments, we set the period of PM to the sampling period of the positioning devices. The number of positioning devices is large, and they are usually not synchronized, which means that the pre-processing module can produce more than one observation within a sampling period. As a result, the reevaluation frequency of NM is higher than that of PM. (3) Critical Devices Method (CDM): The method we propose in the paper. A computer with Windows XP professional, a 2.66GHz Core2 Duo CPU, and 3.25GB main memory is used to run all experiments.

5.1 Memory Consumption

Since the QHT and $DQHT$ are stored in the main memory, we give a brief analysis on their worst-case memory consumption with respect to the synthetic data.

Each key (query identifier) in QHT has two bitmaps, for certain result and uncertain result respectively. We use an *int* value for each query identifier, which occupies 4 bytes. A 6,250 byte bitmap is enough for representing the 50K moving objects, which is the largest number of objects in the experiment [2]. As a result, each

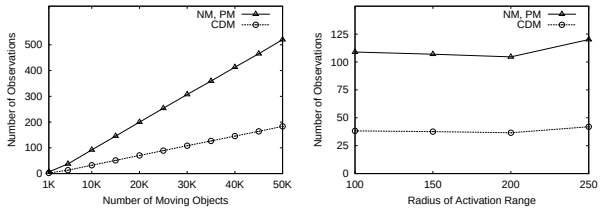
[2]50K also approximates to the peak population in the busiest station in the London Underground motivating example. For continuous monitoring

entry in QHT is 12,504 bytes. For 5K queries, the total size of QHT is 5K·12,504= 62.52M bytes.

The size of $DQHT$ is related to the query range. A bigger query range results in more corresponding critical devices. In the worst case, the range can be the entire indoor space, which makes all positioning devices critical. We use an *int* value for each device identifier and a 1 byte *byte* value to indicate the class of each critical device. If the 5K queries have the whole indoor space as their query ranges, the largest size of each entry in $DQHT$ is (4+5K·(4+1))= 25,004 bytes. For all the 143 devices, the size of $DQHT$ is 143 · 25,004≈3.5M bytes, a modest main memory consumption.

5.2 Workload Reduction

During continuous query updating, different methods use different numbers of observations to update the query results. We quantify a workload by the average number of observations used within each sampling period to update the query results. We use synthetic data in this experiment. Our CDM method is able to reduce the workload significantly, as seen in Figure 7. For both NM and PM, all the observations from the pre-processing are used to reevaluate queries because they are query blind. For CDM, only the observations from critical devices are considered for a query.



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

Figure 7: Workload Reduction

In Figure 7(a), we fix the number of queries at 1K and vary the number of objects from 1K to 50K. The workloads of NM and PM increase markedly with increasing numbers of objects, while CDM performs much more steadily.

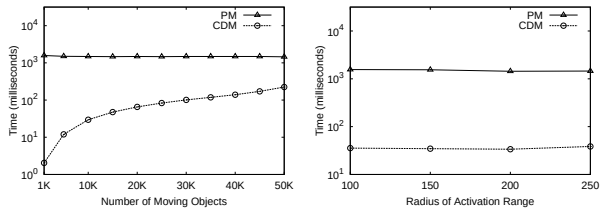
To observe the effect of the activation range, we vary the radius from 100cm to 250cm. The results are reported in Figure 7(b). As the number of observations after pre-processing is affected only little by the varying radius, the workload is also stable. Note that CDM outperforms NM and PM significantly.

5.3 Query Result Update Efficiency

We proceed to compare PM and CDM in terms of the efficiency of the continuous query result update. We consider the average CPU time spent on query result update per sampling unit. We omit NM because it incurs considerably more time by reevaluating each query whenever a new observation arrives. In a real setting, the indoor positioning devices can produce large numbers of raw readings and pre-processing observations, rendering NM infeasible.

Using the synthetic data, we fix the number of queries at 1K and vary the number of moving objects and the activation range. The results are reported in Figure 8. It is seen that CDM is considerably more efficient than PM. In PM, all queries are reevaluated every sampling period. The query processing module fetches its results from corresponding hash tables for all queries. The time used remains constant for varying object counts and activation ranges, approaching 1.5 seconds per sampling period. This indicates that only if the sampling period of positioning devices exceeds 1.5 seconds can the system guarantee that all queries are always reevaluated on

purpose, only online data needs to be stored in the proposed structures. Management of historical data is beyond the scope of this paper.



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

Figure 8: Query Result Update Efficiency

time. In a real setting, however, the sampling unit is usually smaller than 1.5 second [19].

In CDM, query results are updated incrementally only when a new observation comes from a critical device. As a result, continuous query result update consumes much less time than does PM. This indicates that CDM is a practical and efficient solution in a real setting.

The cost of one-time query registration in CDM is almost the same as that of a result update in PM. The cost of un-registering a query is similar to that of a result update in CDM, as it only deletes relevant items from the memory-resident *DQHT* and *QHT*.

To investigate the effect of the deferred query result updates that exploit the maximum speed constraint (MSC), we compare the time difference between CDM with MSC and CDM without MSC.

Since the maximum speed constraint only applies to observations from $CLASS2$ and $CLASS4$ critical devices, we generate a set of queries that concern only $CLASS2$ and $CLASS4$ critical devices. The number of such queries is varied from 500 to 5K. The default values are used for the object number and activation range.

For each query, the minimum indoor walking distance from the query range to each critical device is calculated and recorded during query registration. As these distances are constant, we do not need to calculate them repeatedly during continuous query result update.

The relevant results are reported in Figure 9. The computation cost increases as the number of queries grows for both methods. Although the processing cost of CDM with MSC is higher than that of the CDM without MSC, the total processing time is still quite low, i.e., less than 0.2 seconds per sampling period, and it is still much better than PM (around 1.5 seconds per sampling period; see Figure 8).

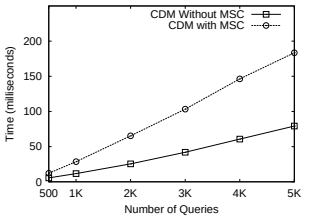


Figure 9: Deferred Query Result Update

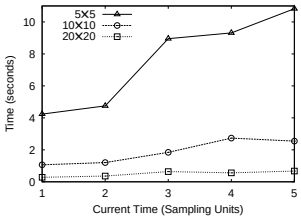


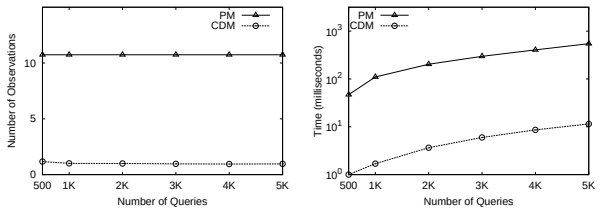
Figure 10: Probabilistic Analysis

Next, we investigate the efficiency of the probabilistic analysis proposed in Section 4.4. We generate a set of queries, each with a room as its range. The Monte Carlo method is employed to compute the relevant areas used in the definitions in Section 4.4. We set the basic unit to 5cm×5cm, 10cm×10cm, and 20cm×20cm, respectively, and the area is measured as the number of units. A smaller unit produces more accurate results while also yielding a higher computational cost, as shown in Figure 10. The probability is reevaluated for every sampling unit. As time passes, the radius of the maximum speed constraint circle increases and thus results

in higher computational cost. For the 20cm×20cm unit setting, which is an acceptable accuracy level for indoor space, the processing time is the lowest, and it remains largely constant.

5.4 Scalability

Finally, we investigate the scalability of PM and CDM by varying the query number from 500 to 5K. We use the real data set in this experiment. We consider both the workload and query update efficiency. The average numbers of used observations for each query per sampling unit are reported in Figure 11(a). Both CDM and PM exhibit near constant performance, with CDM using much fewer observations.



(a) Workload (b) Efficiency

Figure 11: Scalability in Number of Queries

The results on average CPU time for updating the query results per sampling period are reported in Figure 11(b). As the number of queries increases, PM degrades markedly; in contrast, CDM performs is almost unaffected, with only a slightly visible CPU cost increase. The results indicate that CDM is scalable in terms of the number of concurrent queries.

6. RELATED WORK

Various techniques have been proposed for continuous range monitoring queries in the context of outdoor free-moving objects. The basic method to process continuous queries involves periodic reevaluation [6, 15, 16]. The degree of accuracy is largely determined by the reevaluation frequency, which is hard to optimize.

To facilitate query processing, indexes [17, 18] have been proposed for the current locations of moving objects that are modeled as linear functions of time. Alternatives [4, 11, 13] support frequent location updates of moving objects. By differentiating positive and negative updates on the query result, incremental processing reduces the workload in reevaluation [15]. These proposals are not suited for indoor moving objects because they assume that the objects move in Euclidean spaces.

Querying imprecise outdoor location data has been studied recently. Cheng et al. [5] propose a framework to process the probabilistic range query and nearest neighbor query over moving objects. Ishikawa et al. [12] consider queries with imprecise query locations against exact locations. Yiu et al. [14] study probabilistic queries on existentially uncertain data. The techniques in these works, however, are not directly applicable to our indoor setting.

7. CONCLUSION AND FUTURE WORK

This paper addresses incremental and query-aware processing of continuous range monitoring queries against objects moving in symbolic indoor spaces. Based on an indoor positioning device deployment, states of the indoor moving objects are identified, which in turn are utilized to design effective indexing structures for the indoor moving objects. Given a continuous range monitoring query, its critical devices are identified to obtain the initial query result and significantly constrain the search space for future query result updates. Due to the limitations of indoor positioning devices, query results are partitioned into certain and uncertain parts. By taking advantage of the categorization of critical devices, query results are updated in an incremental manner. Exploiting a maximum speed constraint for objects, the paper also offers a probabilistic analysis of the uncertain query results. An experimental study is conducted on both synthetic and real data. The results suggest that our proposal is efficient and scalable.

Several interesting research directions exist. First, it is possible to share query processing cost among concurrent queries so as to reduce the overall system overhead. Critical devices common to multiple queries can be identified and exploited for that purpose. Second, it is of interest to consider other types of monitoring queries, e.g., range and kNN queries that are attached to moving objects. Third, it is 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's work was done when he was a Visiting Scientist at Google Inc.

8. REFERENCES

[1] Transport for London. http://www.tfl.gov.uk/.
[2] P. Bahl and V. Padmanabhan. RADAR: An In-Building RF-Based User Location and Tracking System. In *Proc. INFOCOM*, pp.775–784, 2000.
[3] C. Becker and F. Dürr. On Location Models for Ubiquitous Computing. *Personal Ubiquitous Computing*, 9(1):20–31, 2005.
[4] L. Biveinis, S. Saltenis, C. S. Jensen. Main-Memory Operation Buffering for Efficient R-Tree Update. In *Proc. VLDB*, pp. 591–602, 2007.
[5] R. Cheng, D. V. Kalashnikov, and S. Prabhakar, "Querying imprecise data in moving object environments," *IEEE Trans. Knowl. Data Eng.*, vol. 16, no. 9, pp. 1112–1127, 2004.
[6] C. S. Jensen, D. Lin, and B. C. Ooi. Query and Update Efficient B^+-tree Based Indexing of Moving Objects. In *Proc. VLDB*, pp. 768–779, 2004.
[7] C .S. Jensen, H. Lu, and B. Yang. Graph Model Based Indoor Tracking. In *Proc. MDM*, pp. 122–131, 2009.
[8] S. Feldmann, K. Kyamakya, A. Zapater and Z. Lue, An Indoor Bluetooth-Based Positioning System: Concept, Implementation and Experimental Evaluation, *Proc. ICWN*, pp. 109–113, 2003
[9] J. Hightower and G. Borriello. Location Systems for Ubiquitous Computing. *Computer*, 34:57–66, 2001.
[10] H. Hu, J. Xu, and D. L. Lee. A Generic Framework for Monitoring Continuous Spatial Queries over Moving Objects. In *Proc. SIGMOD*, pp. 479–490, 2005.
[11] D. V. Kalashnikov, S. Prabhakar, and S. E. Hambrusch. Main Memory Evaluation of Monitoring Queries over Moving Objects. *Distributed and Parallel Databases*, 15(2):117–135, 2004.
[12] Y. Ishikawa, Y. Iijima and J. X. Yu, "Spatial Range Querying for Gaussian-Based Imprecise Query Objects", *ICDE*, pp. 676–687, 2009.
[13] M.-L. Lee, W. Hsu, C. S. Jensen, B. Cui, and K. L. Teo. Supporting Frequent Updates in R-trees: A Bottom-up Approach. In *Proc. VLDB*, pp. 608–619, 2003.
[14] M. L. Yiu, N. Mamoulis, X. Dai, Y. F. Tao and M. Vaitis, "Efficient Evaluation of Probabilistic Advanced Spatial Queries on Existentially Uncertain Data", *IEEE Trans. Knowl. Data Eng.*,vol. 16, no. 1, pp. 108-122, 2009.
[15] M. F. Mokbel, X. Xiong, and W. G. Aref. SINA: Scalable Incremental Processing of Continuous Queries in Spatio-temporal Databases. In *Proc. SIGMOD*, pp. 623–634, 2004.
[16] S. Prabhakar, Y. Xia, D. V. Kalashnikov, W. G. Aref, and S. E. Hambrusch. Query Indexing and Velocity Constrained Indexing: Scalable Techniques for Continuous Queries on Moving Objects. *IEEE Trans. Computers*, 51(10):1124–1140, 2002.
[17] S. Saltenis, C. S. Jensen, S. T. Leutenegger, and M. A. Lopez. Indexing the Positions of Continuously Moving Objects. In *Proc. SIGMOD*, pp. 331–342, 2000.
[18] Y. Tao, D. Papadias, and J. Sun. The TPR*-tree: An Optimized Spatio-temporal Access Method for Predictive Queries. In *Proc. VLDB*, pp. 790–801, 2003.
[19] R. Want. *RFID Explained: A Primer on Radio Frequency Identification Technologies*. Morgan and Claypool, 2006.