Scalable Continuous Range Monitoring of Moving Objects in Symbolic Indoor Space

Bin Yang\textsuperscript{1,2}  Hua Lu\textsuperscript{2}  Christian S. Jensen\textsuperscript{1}

\textsuperscript{1}Department of Computer Science, Aalborg University, Denmark
\textsuperscript{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, wayfinding, 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 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
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](image)

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 \((\text{deviceID}, \text{objectID}, t)\) states that object \(\text{objectID}\) is detected by device \(\text{deviceID}\) 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 \(\text{deviceID}_1\) and \(\text{deviceID}_2\) is assigned to a device \(\text{deviceID}_{12}\). An object seen by \(\text{deviceID}_1\), but not \(\text{deviceID}_2\), is then in the non-intersecting part of the range of \(\text{deviceID}_2\). 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 \((\text{deviceID}, \text{objectID}, t, \text{flag})\). Here, \(\text{flag} = \text{ENTER}\) indicates that the object is entering the device’s activation range; \(\text{flag} = \text{LEAVE}\) indicates the object is leaving the range. The \(\text{deviceID}\) can be that of a virtual device. Unless explicitly stated otherwise, this applies to all \(\text{deviceID}\)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, \(\text{deviceID}_{21}\) cannot tell whether an observed object enters or leaves cell \(\text{c}_{21}\). Note that \(\text{deviceID}_{11}\), \(\text{deviceID}_{11'}\), and \(\text{deviceID}_{12}\) 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 \(\text{deviceID}_{11}\) and \(\text{deviceID}_{11'}\) in Figure 1.

Next, presence devices (PR) simply sense the presence of objects in their ranges. These are exemplified by \(\text{deviceID}_{30}\) in Figure 1.

A Devices mapping maintains the information on the positioning devices:

\[
\text{Devices} : \Sigma_{\text{devices}} \rightarrow \{(\text{ActRange}, \text{TYPE})\},
\]

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

To facilitate object tracking and querying, a deployment graph \(G = (C, E, \Sigma_{\text{devices}}, \ell_E)\) 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, \(\ell_E : E \rightarrow 2^{\Sigma_{\text{devices}}}\) 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 [7]. The deployment graph of Figure 1 is shown in Figure 2, where the label \(D_i\) indicates the positioning device \(\text{deviceID}_i\).

![Figure 2: Positioning Device Deployment Graph](image)

Each cell created by a reader deployment corresponds to one or more rooms. For example, cell \(\text{c}_{10}\) is mapped to rooms 10 and 14 because an object can go between these rooms without being observed. A mapping \(\text{Cells} : C \rightarrow 2^{\Sigma_{\text{rooms}}}\) maintains this relationship, where \(\Sigma_{\text{rooms}}\) is the domain of room identifiers.
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 in the cell \(G.ℓ \setminus \Sigma_d\) (d) until it is again seen \(^1\). Therefore, its state changes from active to deterministic. In our running example, if an object leaves device10, it must enter c10. If an object leaves a directed positioning 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 device11, and then device11, it must enter c11.

In contrast, if an object leaves an undirected positioning device, the object can be in either of the cells in \(G.ℓ \setminus \Sigma_d\) (d). Therefore, its state changes from active to nondeterministic. For example, if a moving object leaves device12, it can be in either c10 or c12.

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 positioning 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](image)

\(G.ℓ \setminus \Sigma_d\) is the reverse function of \(G.ℓ\) introduced in Section 2.2. For simplicity, we throughout the paper use \(G.ℓ \setminus \Sigma_d\) (d) to denote \(G.ℓ \setminus \Sigma_d\) (D), where \(D \subseteq \Sigma_d\). 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 \(O_{\text{indoor}}\) 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:

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

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

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

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

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

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

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

Here \(\text{STATE}\) denotes the object’s current state; \(t\) is the start time of the state; \(\text{IDSet}\) 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, \(\text{IDSet}\) is a singleton set consisting of a device identifier. If the state is deterministic, \(\text{IDSet}\) is a singleton set consisting of a cell identifier. If the state is nondeterministic, \(\text{IDSet}\) 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 \(\text{flag}\) value.

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

1: \(\text{IDSet } s\text{Set} \leftarrow \emptyset;\)
2: if \(\text{O.flag } = \text{ENTER}\) then
3: \(\text{sSet} \leftarrow \text{OHT[O.objectID], IDSet;}\)
4: if \(\text{OHT[O.objectID].STATE } = \text{Active}\) then
5: for the single element \(c\) in \(s\text{Set}\) do
6: \(\text{Delete O.objectID from DHT[c];}\)
7: else if \(\text{OHT[O.objectID].STATE } = \text{Deterministic}\) then
8: for the single element \(c\) in \(s\text{Set}\) do
9: \(\text{Delete O.objectID from CDHT[c];}\)
10: else
11: for each element \(c\) in \(s\text{Set}\) do
12: \(\text{Delete O.objectID from CNHT[c];}\)
13: \(\text{Add O.objectID to DHT[O.deviceID];}\)
14: \(\text{OHT[O.objectID]} \leftarrow (\text{Active, } t, \text{IDSet; O.objectID});\)
15: else
16: \(\text{Delete O.objectID from DHT[O.deviceID];}\)
17: \(\text{sSet} \leftarrow G.ℓ \setminus \Sigma_{\text{deviceID}};\)
18: if \(\text{Device}'s(\text{O.deviceID})\).TYPE = \text{UP}\) then
19: \(\text{OHT[O.objectID]} \leftarrow \text{(Nondeterministic, } t, \text{sSet; )}\)
20: for each element \(c\) in \(s\text{Set}\) do
21: \(\text{Add O.objectID to CNHT[c];}\)
22: else
23: \(\text{OHT[O.objectID]} \leftarrow \text{(Deterministic, } t, \text{sSet; )}\)
24: for each element \(c\) in \(s\text{Set}\) do
25: \(\text{Add O.objectID to CDHT[c];}\)

For an \text{ENTER} record, if the object’s previous state is active, it is deleted from the corresponding device’s \(\text{DHT}\) (lines 4–6). If its previous state is deterministic, it is deleted from the corresponding cell’s \(\text{CDHT}\) (lines 7–9). Otherwise, its previous state is nondeterministic, and the object is deleted from all corresponding cells’ \(\text{CNHTs}\) (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 \( G.L_c \) (lines 17). If the object leaves a UP device, its state becomes nondeterministic, and the object is added into all the corresponding cells' CNHTs (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.

\[
\forall t \in [t_s, t_e]; \exists o \in \text{CRMQ}(R)(M) \iff o \in M \land \text{pos}_M(o, t) \in R,
\]

where \( M \) indicates all the objects moving in the indoor space; \( \text{pos}_M \) 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.

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 of the CRMQ. For example, the range of a CRMQ \( \text{query}_{11} \) is shown as a dashed rectangle in Figure 5. The active objects in \( \text{device}_{13} (\text{device}_{16}) \) 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 \( \cap \) return the intersection of two argument regions. If \( x \cap y \neq \emptyset \) and neither \( x \cap y \), \( x \cap y \), \( x \cap y \), we say that \( x \) intersects with \( y \) if \( x \cap y \neq \emptyset \) and \( x \cap y \), 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 \( \text{query}_{11} \) in Figure 5. The deterministic (nondeterministic) objects in \( c_{12} \) 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 \( \text{query}_{11} \).

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 \( \text{query}_{12} \) whose range is rooms 12 and 13. The query range covers both \( c_{12} \) and \( c_{13} \). If an object leaves \( \text{device}_{16} \), 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 $C_{ic} = \{e | e \cap R \neq \emptyset\}$. In order to guarantee the accuracy of the query result, we introduce another extended set of cells $C_{ic} = \{c | (c, c') \in G.ℓ, c' \in C_{ex}\}$, which contains the neighbor cells of $C_{ic}$. For example, for query $q_2$ 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 $d' = \text{Devices}(d).\text{ActRange}$. To improve the query processing, we categorize all critical devices into five classes.

$$\text{CLASS}1 = \{d|d' \cap R = d'; \forall e \in G.ℓ(d) \cap R = e\}$$

$$\text{CLASS}2 = \{d|d' \cap R = d'; \exists e \in G.ℓ(d) \cap R = e; e \cap R \neq \emptyset\}$$

$$\text{CLASS}3 = \{d|d' \cap R \neq d'; d' \cap R \neq \emptyset\}$$

$$\text{CLASS}4 = \{d|d' \cap R = \emptyset; \exists e \in G.ℓ(d) \cap C_{ex}\}$$

$$\text{CLASS}5 = \{d|d' \cap R = \emptyset; \forall e \in G.ℓ(d) \cap C_{ex}; \exists e \in G.ℓ(d) \cap C_{ex}\}$$

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 $q_2$ in Figure 5, $\text{device}_{16}$ is a CLASS1 device.

The activation range of a CLASS2 device intersects with the query range $R$. For example, $\text{device}_{13}$ is a CLASS2 devices of query $q_1$.

The activation range of a CLASS3 device is disjoint with the query range $R$ and at least one of its corresponding cells is in $C_{ex}$. For example, $\text{device}_{10}$ is a CLASS3 device of query $q_1$.

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, $\text{device}_{10}$ is a CLASS5 critical device of query $q_2$.

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:

$$\text{DQHT}[\text{device ID}] = \{(\text{query ID}, \text{CLASS})\},$$

where $\text{device ID}$ indicates a device, $\text{query ID}$ indicates a query that has the device as a critical device, and $\text{CLASS}$ 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 $\text{Query Hash Table (QHT)}$ is created in main memory. It maps a query identifier to the query’s results:

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

where $\text{CR}$ is the certain result and $\text{UR}$ 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 $\text{CLASS}3$ (lines 15–16). After that, $\text{CLASS}4$ 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 $\text{DHT}$ are added to the certain result (lines 26–27). For each device in the intersected device set, all the corresponding active objects from $\text{DHT}$ are added to the uncertain result (lines 28–29). For each cell in the covered cell set $C_{ic}$, all the corresponding deterministic objects from $\text{CDHT}$ 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 $\text{OHT}[o].\text{IDSet}$ are in $C_{ic}$, the object is added to the certain result. Otherwise, the object is added to the uncertain result (lines 32–37). If $C_{ic}$ has only one cell, all the corresponding nondeterministic objects from $\text{CNHT}$ are added to the uncertain result (lines 38–39). For each cell in the intersected cell set, both deterministic objects from $\text{CDHT}$ and nondeterministic objects from $\text{CNHT}$ 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 1 register** (*Range R, DeploymentGraph G*)

1. deviceSet \( D_e \leftarrow \emptyset; \) \( D_{uc} \leftarrow \emptyset; \)
2. cellSet \( C_e \leftarrow \emptyset, \) \( C_{uc} \leftarrow \emptyset; \)
3. objectSet \( R_e \leftarrow \emptyset, \) \( R_{uc} \leftarrow \emptyset; \)
4. CriticalDeviceList(\( deviceID; \) CLASS) \( cd \leftarrow \emptyset; \)
5. Generate a new identifier queryID for the query;
6. \( D_e \leftarrow \) Devices that are covered by \( R; \)
7. \( D_{uc} \leftarrow \) Devices that intersect with \( R; \)
8. \( C_e \leftarrow \) Cells which are covered by \( R; \)
9. \( C_{uc} \leftarrow \) Cells that intersect with \( R; \)
10. for each device \( d \) in \( D_{uc} \)
11. if all the cells in \( G_1G_2^{-1}(d) \) are in \( C_e \)
12. Add \((d, \text{CLASS1})\) to \( C_e \);
13. else if one of the cells in \( G_1G_2^{-1}(d) \) is in \( C_{uc} \)
14. Add \((d, \text{CLASS2})\) to \( C_{uc} \);
15. for each device \( d \) in \( D_{uc} \)
16. Add \((d, \text{CLASS3})\) to \( C_{uc} \);
17. for each edge \( e \) in \( G \)
18. if \((c_e \cup C_{uc}) \cap e \neq \emptyset \) AND \((C_e \cup C_{uc}) \cap e \neq (C_e \cup C_{uc})\)
19. if \(G_e \in C\) \( \notin \emptyset \) or device.ID \( I \)
20. Add \((G_e \in C), \text{CLASS4})\) to \( C_e \);
21. \( C_{uc} \leftarrow C_e \cup e \notin (C_e \cup C_{uc}); \)
22. for each edge \( e \) in \( G \)
23. if \( C_{uc} \cup e \neq \emptyset \)
24. if \(G_e \in C\) \( \notin \emptyset \) or device.ID \( I \)
25. Add \((G_e \in C), \text{CLASS5})\) to \( C_e \);
26. for each device \( d \) in \( D_{uc} \)
27. \( R_e \leftarrow R_{uc} \cup \text{DHT} \{d\}; \)
28. for each device \( d \) in \( D_{uc} \)
29. \( R_{uc} \leftarrow R_{uc} \cup \text{DHT} \{d\}; \)
30. for each cell \( c \) in \( C_e \)
31. \( R_e \leftarrow R_e \cup \text{DHT} \{c\}; \)
32. if \( \forall c \in C \) \( \exists t \) then
33. for each nondeterministic object \( o \) in \( C_e \)
34. if \( D Q H T \{o\} \cap D \text{Set} \subset C_e \)
35. Add \( o \) into \( R_e \); \)
36. else
37. Add \( o \) into \( R_{uc} \);
38. else
39. \( R_{uc} \leftarrow R_{uc} \cup \text{CNHT} \{e\}; \)
40. for each cell \( c \) in \( C_{uc} \)
41. \( R_e \leftarrow R_e \cup \text{DHT} \{c\}; \)
42. \( R_{uc} \leftarrow R_{uc} \cup \text{CNHT} \{c\}; \)
43. \( Q HT(\text{queryID}) \leftarrow (R_e, R_{uc}); \)
44. for each item \( i \) in \( cd \) do
45. Add \((\text{queryID}, \text{a.CLASS})\) into \( D Q H T \{a.deviceID\}; \)

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>
<thead>
<tr>
<th>Table 1: Query Updates w.r.t. Critical Devices</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ENTER</strong></td>
</tr>
<tr>
<td>CLASS1</td>
</tr>
<tr>
<td>CLASS2</td>
</tr>
<tr>
<td>CLASS3</td>
</tr>
<tr>
<td>CLASS4</td>
</tr>
<tr>
<td>CLASS5</td>
</tr>
</tbody>
</table>

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 system 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 queryID \( o \) in Figure 5. According to Table 1, after an
object \( o \) leaves a CLASS2 critical device deviceID \( o \), 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 deviceID \( o \) to current time be \( \Delta t \).
The longest possible distance \( o \) can move from the boundary of the
device’s 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 deviceID \( o \)
as the center, and the radius of deviceID \( o \) 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 \( \Delta t \).

Assuming that function \( \text{minDist}(\text{deviceID}, R) \) 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
\( \Delta t \) is determined as \( \text{minDist}(\text{deviceID}, R) \cdot V_{max} \). After
the time period \( \Delta t \), 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., deviceID \( o \) for queryID \( o \) in Figure 5, \( o \) is added
to the uncertain result of the query. As a matter of fact, only after
some time \( \Delta t=\text{minDist}(\text{deviceID}, R) \cdot V_{max} \), can the object \( o \)
possibly enter the query range. As shown in Figure 5, the function
\( \text{minDist}(\text{deviceID}, \text{range}) \) returns the minimum indoor walking
distance, which is \( R_{3}+R_{4} \) rather than the Euclidean distance.
to the left in Figure 6. The constraint circle \( R \) denotes the accessible intersection between the maximum speed constraint circle and query range \( \Theta \). The probability \( \Theta \) is defined as:

\[
\frac{\text{Area}(c_i \setminus \text{Circle}(o, V))}{\text{Area}(c_i)},
\]

where \( c_i \) is defined as the object \( o \) is in the shaded region in Figure 6.

### 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 \( o \) is the uncertain set, the positioning device \( C_d \) indicates the probability that \( o \) contains all the critical devices. Each time when the query is to be updated, the corresponding record should be deleted from \( D \).

To incorporate deferred query result updates, the query update module needs slight changes as follows. For an \textit{ENTER} observation, if the object in \( D \), the corresponding record should be removed from \( D \). For a \textit{LEAVE} observation, if the critical device is a \textit{CLASS2} device, a \( C_2U \) update record is created and inserted into \( D \) with the corresponding future update time. If the device is a \textit{CLASS4} device, a \( N_2U \) update record is created and inserted into \( D \).

#### 4.4.1 Probabilities for Active Objects

Recall from Section 4.3.2 that all the objects in the activation range of a \textit{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:

\[
\text{prob}(o \in R) = \frac{\text{Area}(\text{Device}(d) \cap \Theta)}{\text{Area}(\text{Device}(d) \setminus \text{ActRange})}.
\]

In Figure 5, \( \text{Device}(d) \) is a \textit{CLASS3} device for \textit{query1}, and the probability for an active object in \( \text{Device}(d) \) to be in the query range is calculated as:

\[
\frac{\text{Area}(\text{Device}(d) \cap \Theta)}{\text{Area}(\text{Device}(d) \setminus \text{ActRange})}.
\]

#### 4.4.2 Probabilities for Inactive Objects

As described in Section 4.3.3, after leaving a \textit{CLASS2}, \textit{CLASS3}, or \textit{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 \( o \in D \) to indicate that the inactive object \( o \) has most recently left critical device \( d \). Let the set \( CD234 \) contain all the \textit{CLASS2}, \textit{CLASS3}, and \textit{CLASS4} critical devices for a given query. The probability that an inactive object \( o \) is in query range \( R \) is defined as:

\[
\text{prob}(o \in R) = \sum_{d \in CD234} \text{prob}(o \in D \cap R | o \in D)
\]

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

After leaving the positioning device \( d \), the object should enter exactly one specific cell. The probability of the object being in this cell is 1. If \( d \) is a \textit{UP} device, the object must enter exactly one specific cell. The probability \( \text{prob}(o \in D) \) 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:

\[
\text{prob}(o \in D | c_i, H) = \begin{cases} 
\frac{\text{Bound}(d, c_i)}{\text{Bound}(d)} & \text{if } d \text{ is UP} \\
1 & \text{otherwise}
\end{cases}
\]

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

\[
\text{prob}(o \in D | c_i, H) = \begin{cases} 
\frac{\text{Area}(c_i \cap \text{Circle}(d, V))}{\text{Area}(c_i)}, & c_i \cap R \neq \emptyset \\
0, & \text{otherwise}
\end{cases}
\]

Refer to the example in Figure 6. Assume that the assumption \( H \) indicates that object \( o \) most recently left device12 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 \( V_{\text{max}} \) and let the time span from the current time to be \( \Delta T \). Then \( R_1 = \Delta T \cdot V_{\text{max}} \). 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 \( \Delta T \).

Next, let the minimum indoor walking distance from the door to the boundary of device12 be \( \text{minDist}(device12, \text{door}) \). If \( \Delta T < \text{minDist}(device12, \text{door}) \), the object cannot enter room 14. Here, \( c_{10} \textrm{Circle} \) denotes the accessible intersection between the maximum speed constraint circle \( R_1 \) and the part of cell \( c_{10} \) within room 10. Continuing, \( c_{10} \cap \text{Circle}(d, V) \) is the intersection between the accessible region \( c_{10} \) and query range \( R \), shown as the shaded region to the left in Figure 6.
Next, we define the relationship \( \Lambda \). 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\( _{33} \) 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\( _{33} \) in the same cell can be a direct or an indirect part depending on the device assumed.

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

\[
c \text{Circle} = \{DP(c, d) \cap \text{Circle}(t)\} \cup \{IP(c, d) \cap \text{Circle}(t')\}
\]

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 \( \Delta T \)). 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., \( t' = \frac{l}{V_{\text{max}}} \).

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 \( \text{device}_{10} \) and \( \text{device}_{12} \) 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 the same 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.

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 pre-processing 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. As a result, each

\(^{50K}\)also approximates to the peak population in the busiest station in the London Underground motivating example. For continuous monitoring

<table>
<thead>
<tr>
<th>Number of Objects</th>
<th>1K, 10K, 20K, 30K, 40K, 50K</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activation Range</td>
<td>100, 150, 200, 250 (cm)</td>
</tr>
<tr>
<td>Number of Queries</td>
<td>500, 1K, 2K, 3K, 4K, 5K</td>
</tr>
</tbody>
</table>
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 $inr$ 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 $143 \cdot 25,004=3.5M$ 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.

![image](image1.png)

(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.

![image](image2.png)

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

Figure 8: Query Result Update Efficiency

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).

![image](image3.png)

Figure 9: Deferred Query Re-

Figure 10: Probabilistic Analy-

sis

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 \times 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.

**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 \( k \)NN 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