Manage the Data from Indoor Spaces: Models, Indexes & Query Processing

Huan Li

Database Laboratory, Zhejiang University

lihuancs@zju.edu.cn

March 17, 2016

Overview

1. 1. Outlines
2. 2. Indoor Space Models & Applications
3. 3. Indoor Data Cleansing
4. 4. Indoor Movement Analysis
5. 5. Appendix

Aims

- To give a brief review introduction to *indoor data management techniques*.
- To review a series of works in this field, including their proposed *models*, *indexes* and *algorithms*.
- To discuss how to bring those advanced theoretical contents into practice.

About This Work...

Scalable Continuous Range Monitoring of Moving Objects in Symbolic Indoor Space. [3]
B. Yang, H. Lu, and C. S. Jensen.

- Published in *CIKM' 2009*.
- Application: continuously monitor indoor moving objects for space use analysis or security purposes.
- An incremental, query-aware continuous range query processing technique for objects moving in indoor space.
- Use maximum speed constraint on object movement to refine the uncertain results.

Motivation

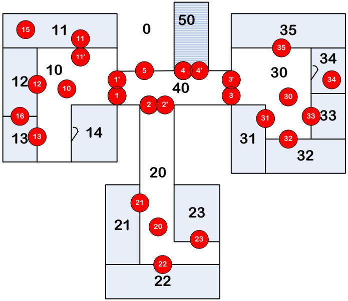
- People spend much time in indoor spaces.
- Indoor spaces are becoming increasingly larger and complex.
  - E.g., London Underground, 268 stations, 408 kilometers of network, +4 million daily passengers.
- Indoor monitoring of people can help support.
  - space use analysis
  - security purposes

Preliminaries: Indoors vs. Outdoors

- Modeling of indoor spaces do not assume
  - Euclidean space. (since obstacles render movement more constrained)
  - Spatial network. (since indoor movement is less constrained than movements in polylines)
- Instead indoor spaces are characterized by entities.
  - Doors, rooms, hallways, staircase, etc.
- **Symbolic models** are more suitable.
- *GPS* and *cellular tracking* do not work indoors.
- Sensing devices are used to detect objects within their activation range, e.g., RFID readers or Bluetooth hotspots.

Positioning Devices Deployment Graph

- Two types of positioning devices
  - Partitioning Device – *undirected* (**UP**), e.g., d_{21} – *directed* (**DP**), e.g., d_{11} and $d_{11'}$
  - Presence Device – (**PR**)
- Note an indoor space is partitioned into *activation ranges* and *cells*



Deployment Graph

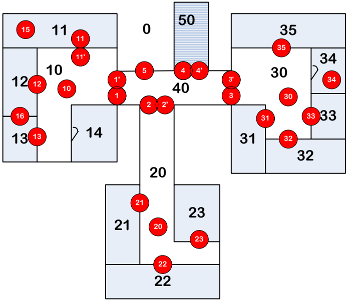
- $G = \{C, E, \Sigma_{devices}, l_E\}$
- C: the set of cells
- E: the set of edges, $\{c_i, c_j\}$ where $c_i, c_j \in C$
- $\Sigma_{devices}$: a mapping from $deviceID$ to activation range and type
- l_E maps an edge to a set of positioning devices, i.e., $E \rightarrow 2^{\Sigma_{devices}}$

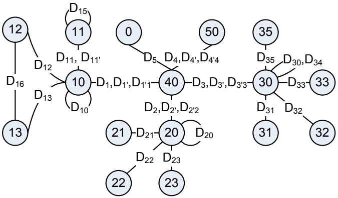
Positioning Devices Deployment Graph

- Two types of positioning devices
  - Partitioning Device – *undirected* (**UP**), e.g., d_{21} – *directed* (**DP**), e.g., d_{11} and $d_{11'}$
  - Presence Device – (**PR**)
- Note an indoor space is partitioned into *activation ranges* and *cells*

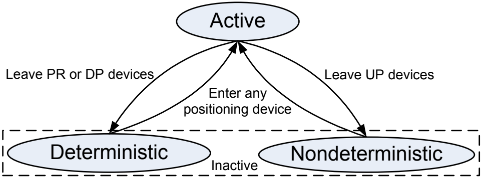
Deployment Graph

- $G = \{C, E, \Sigma_{devices}, l_E\}$
- C: the set of cells
- E: the set of edges, $\{c_i, c_j\}$ where $c_i, c_j \in C$
- $\Sigma_{devices}$: a mapping from $deviceID$ to activation range and type
- l_E maps an edge to a set of positioning devices, i.e., $E \rightarrow 2^{\Sigma_{devices}}$





States of Indoor Moving Objects



- An object is in an **active state** when it is inside the activation range of a positioning device.
- Otherwise the object is in an **inactive state**
- When an object is in the inactive state it is
  - **nondeterministic** if it can be in more than one cell
  - **deterministic** if it is in one specific cell

Indexing Indoor Moving Objects

The proposed indexing scheme uses 4 hash tables

Indexing Indoor Moving Objects

The proposed indexing scheme uses 4 hash tables

Device Hash Table(DHT) maps each device to a set of active objects:

Indexing Indoor Moving Objects

The proposed indexing scheme uses 4 hash tables

Device Hash Table(DHT) maps each device to a set of active objects:

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

Indexing Indoor Moving Objects

The proposed indexing scheme uses 4 hash tables

Device Hash Table(DHT) maps each device to a set of active objects:

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

Cell Deterministic Hash Table(CDHT) maps each cell to a set of deterministic objects:

Indexing Indoor Moving Objects

The proposed indexing scheme uses 4 hash tables

Device Hash Table(DHT) maps each device to a set of active objects:

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

Cell Deterministic Hash Table(CDHT) maps each cell to a set of deterministic objects:

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

Indexing Indoor Moving Objects

The proposed indexing scheme uses 4 hash tables

Device Hash Table(DHT) maps each device to a set of active objects:

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

Cell Deterministic Hash Table(CDHT) maps each cell to a set of deterministic objects:

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

Cell Nondeterministic Hash Table(CNHT) maps each cell to a set of nondeterministic objects:

Indexing Indoor Moving Objects

The proposed indexing scheme uses 4 hash tables

Device Hash Table(DHT) maps each device to a set of active objects:

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

Cell Deterministic Hash Table(CDHT) maps each cell to a set of deterministic objects:

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

Cell Nondeterministic Hash Table(CNHT) maps each cell to a set of nondeterministic objects:

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

Indexing Indoor Moving Objects

The proposed indexing scheme uses 4 hash tables

Device Hash Table(DHT) maps each device to a set of active objects:

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

Cell Deterministic Hash Table(CDHT) maps each cell to a set of deterministic objects:

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

Cell Nondeterministic Hash Table(CNHT) maps each cell to a set of nondeterministic objects:

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

Object Hash Table(OHT) maps objects to their current data(state, time, cell(s) the object can be in)

Indexing Indoor Moving Objects

The proposed indexing scheme uses 4 hash tables

Device Hash Table(DHT) maps each device to a set of active objects:

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

Cell Deterministic Hash Table(CDHT) maps each cell to a set of deterministic objects:

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

Cell Nondeterministic Hash Table(CNHT) maps each cell to a set of nondeterministic objects:

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

Object Hash Table(OHT) maps objects to their current data(state, time, cell(s) the object can be in)

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

Indexing Indoor Moving Objects

The proposed indexing scheme uses 4 hash tables

Device Hash Table(DHT) maps each device to a set of active objects:

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

Cell Deterministic Hash Table(CDHT) maps each cell to a set of deterministic objects:

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

Cell Nondeterministic Hash Table(CNHT) maps each cell to a set of nondeterministic objects:

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

Object Hash Table(OHT) maps objects to their current data(state, time, cell(s) the object can be in)

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

RFID Deployment Graph Construction

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

1: IDSet $sSet \leftarrow \emptyset$;
2: **if** $O.flag = ENTER$ **then**
3: $sSet \leftarrow OHT[O.objectID].IDSet$;
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: $OHT[O.objectID] \leftarrow (Active, O.t, \{O.deviceID\})$;
15: **else**
16: Delete $O.objectID$ from $DHT[O.deviceID]$;
17: $sSet \leftarrow G.\ell_E^{-1}(O.deviceID)$;
18: **if** $Devices(O.deviceID).TYPE$ = UP **then**
19: $OHT[O.objectID] \leftarrow (Nondeterministic, O.t, sSet)$;
20: **for** each element c in $sSet$ **do**
21: Add $O.objectID$ to $CNHT[c]$;
22: **else**
23: $OHT[O.objectID] \leftarrow (Deterministic, O.t, sSet)$;
24: **for** the single element c in $sSet$ **do**
25: Add $O.objectID$ to $CDHT[c]$;

1. Line 1: reset $IDSet$

RFID Deployment Graph Construction

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];
```

1. Line 1: reset $IDSet$
2. Lines 2–12: $O.flag$ is ENTER so check the object's previous state. Remove O from the corresponding table according its previous state

RFID Deployment Graph Construction

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

1: IDSet $sSet \leftarrow \emptyset$;
2: **if** $O.flag = ENTER$ **then**
3: $sSet \leftarrow OHT[O.objectID].IDSet$;
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: $OHT[O.objectID] \leftarrow (Active, O.t, \{O.deviceID\})$;
15: **else**
16: Delete $O.objectID$ from $DHT[O.deviceID]$;
17: $sSet \leftarrow G.\ell_E^{-1}(O.deviceID)$;
18: **if** $Devices(O.deviceID).TYPE$ = UP **then**
19: $OHT[O.objectID] \leftarrow (Nondeterministic, O.t, sSet)$;
20: **for** each element c in $sSet$ **do**
21: Add $O.objectID$ to $CNHT[c]$;
22: **else**
23: $OHT[O.objectID] \leftarrow (Deterministic, O.t, sSet)$;
24: **for** the single element c in $sSet$ **do**
25: Add $O.objectID$ to $CDHT[c]$;

1. Line 1: reset $IDSet$
2. Lines 2–12: $O.flag$ is ENTER so check the object's previous state. Remove O from the corresponding table according its previous state
3. Lines 13–14: add O to table of active objects (DHT), and update O's in the objects' table (OHT)

RFID Deployment Graph Construction

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

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

1. Line 1: reset $IDSet$
2. Lines 2–12: $O.flag$ is ENTER so check the object's previous state. Remove O from the corresponding table according its previous state
3. Lines 13–14: add O to table of active objects (DHT), and update O's in the objects' table (OHT)
4. Lines 16–17: $O.flag$ is LEAVE so remove the object from DHT. Get the possible cells that O can move to

RFID Deployment Graph Construction

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

1: IDSet $sSet \leftarrow \emptyset$;
2: **if** $O.flag$ = $ENTER$ **then**
3:   $sSet \leftarrow OHT[O.objectID].IDSet$;
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:   $OHT[O.objectID] \leftarrow (Active, O.t, \{O.deviceID\})$;
15: **else**
16:   Delete $O.objectID$ from $DHT[O.deviceID]$;
17:   $sSet \leftarrow G.\ell_E^{-1}(O.deviceID)$;
18:   **if** $Devices(O.deviceID).TYPE$ = UP **then**
19:     $OHT[O.objectID] \leftarrow (Nondeterministic, O.t, sSet)$;
20:     **for** each element c in $sSet$ **do**
21:       Add $O.objectID$ to $CNHT[c]$;
22:   **else**
23:     $OHT[O.objectID] \leftarrow (Deterministic, O.t, sSet)$;
24:     **for** the single element c in $sSet$ **do**
25:       Add $O.objectID$ to $CDHT[c]$;

1. Line 1: reset $IDSet$
2. Lines 2–12: $O.flag$ is ENTER so check the object's previous state. Remove O from the corresponding table according its previous state
3. Lines 13–14: add O to table of active objects (DHT), and update O's in the objects' table (OHT)
4. Lines 16–17: $O.flag$ is LEAVE so remove the object from DHT. Get the possible cells that O can move to
5. Lines 18–25: if the device is undirected, set O in OHT and add O to CNHT for the cells in sSet, else apply the same to CDHT

Continuous Range Monitoring: Query Definition

- A *Continuous Range Monitoring Query* (CRMQ)
  - takes an **indoor spatial range** R as parameter
  - keeps reporting the objects when it is registered for a certain time frame $[t_s, t_e]$
- The **query result** $\mathcal{M}$ – the set of moving objects in R - is maintained as follows:

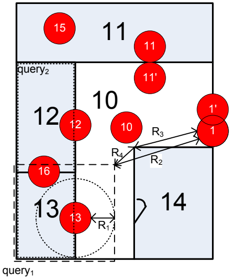
  $$\forall t \in [t_s, t_e] : o \in CRMQ[R](\mathcal{M}) \Leftrightarrow o \in \mathcal{M} \wedge pos_{\mathcal{M}}(o, t) \in R$$

  where $pos_{\mathcal{M}}$ is a function that can determine the position of object o at time t
- Multiple monitoring queries may coexist

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). Use a *Device Query Hash Table* (DQHT) to record the relationships:

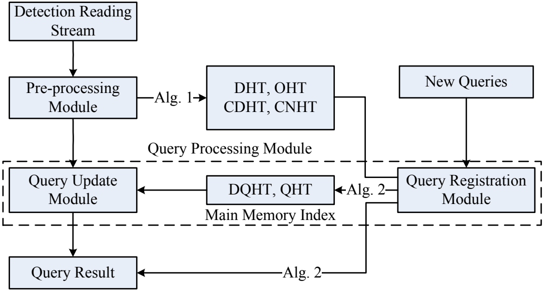
$$DQHT[deviceID] = \{(queryID, CLASS)\}$$



- CLASS1 – Device is fully covered in R along with cells, e.g., $(device_{16}, query_2)$
- CLASS2 – Device is fully covered but corresponding cells are not, e.g., $(device_{13}, query_1)$
- CLASS3 – Device intersects with the query range R, e.g., $(device_{16}, query_1)$
- CLASS4 – Device is disjoint from R and at least one of its corresponding cells in $C_{ic} = \{c | c \sqcap R \neq \varnothing\}$, e.g., $(device_1, query_1)$
- CLASS5 – Device is disjoint from R and at least one of its corresponding cells in $C_{ex} = \{c | \{c, c'\} \in G.E, c' \in C_{ic}\}$, but none of them are in C_{ic}, e.g., $(device_{10}, query_2)$

Query Registration

- To handle concurrent CRMQs, a *Query Hash Table* is created hold the 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
- Overview



Query Registration Algorithm (I)

Algorithm 2 register (Range R, DeploymentGraph G)

1: deviceSet $D_c \leftarrow \emptyset$, $D_{uc} \leftarrow \emptyset$;
2: cellSet $C_c \leftarrow \emptyset$, $C_{uc} \leftarrow \emptyset$, $C_{ex} \leftarrow \emptyset$;
3: objectSet $R_c \leftarrow \emptyset$, $R_{uc} \leftarrow \emptyset$;
4: CriticalDeviceList(deviceID, CLASS) $cd \leftarrow \emptyset$;
5: Generate a new identifier $queryID$ for the query;
6: $D_c \leftarrow$ Devices that are covered by R;
7: $D_{uc} \leftarrow$ Devices that intersect with R;
8: $C_c \leftarrow$ Cells which are covered by R;
9: $C_{uc} \leftarrow$ Cells that intersect with R;
10: **for** each device d in D_c **do**
11:     **if** all the cells in $G.\ell_E^{-1}(d)$ are in C_c **then**
12:         Add (d, $CLASS1$) to cd;
13:     **else if** one of the cells in $G.\ell_E^{-1}(d)$ 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** $(C_c \cup C_{uc}) \cap e \neq \emptyset$ AND $(C_c \cup C_{uc}) \cap e \neq (C_c \cup C_{uc})$ **then**
19:         **if** $G.\ell_E(e) \notin cd.deviceID$ **then**
20:             Add ($G.\ell_E(e)$, $CLASS4$) to cd;
21:         $C_{ex} \leftarrow C_{ex} \cup e \setminus (C_c \cup C_{uc})$;
22: **for** each edge e in G **do**
23:     **if** $C_{ex} \cap e \neq \emptyset$ **then**
24:         **if** $G.\ell_E(e) \notin cd.deviceID$ **then**
25:             Add ($G.\ell_E(e)$, $CLASS5$) to cd;

1. Lines 1–9: Initialization

Query Registration Algorithm (I)

Algorithm 2 register (Range R, DeploymentGraph G)

1: deviceSet $D_c \leftarrow \emptyset$, $D_{uc} \leftarrow \emptyset$;
2: cellSet $C_c \leftarrow \emptyset$, $C_{uc} \leftarrow \emptyset$, $C_{ex} \leftarrow \emptyset$;
3: objectSet $R_c \leftarrow \emptyset$, $R_{uc} \leftarrow \emptyset$;
4: CriticalDeviceList(deviceID, CLASS) $cd \leftarrow \emptyset$;
5: Generate a new identifier $queryID$ for the query;
6: $D_c \leftarrow$ Devices that are covered by R;
7: $D_{uc} \leftarrow$ Devices that intersect with R;
8: $C_c \leftarrow$ Cells which are covered by R;
9: $C_{uc} \leftarrow$ Cells that intersect with R;
10: **for** each device d in D_c **do**
11:     **if** all the cells in $G.\ell_E^{-1}(d)$ are in C_c **then**
12:         Add (d, $CLASS1$) to cd;
13:     **else if** one of the cells in $G.\ell_E^{-1}(d)$ 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** $(C_c \cup C_{uc}) \cap e \neq \emptyset$ AND $(C_c \cup C_{uc}) \cap e \neq (C_c \cup C_{uc})$ **then**
19:         **if** $G.\ell_E(e) \notin cd.deviceID$ **then**
20:             Add ($G.\ell_E(e)$, $CLASS4$) to cd;
21:         $C_{ex} \leftarrow C_{ex} \cup e \setminus (C_c \cup C_{uc})$;
22: **for** each edge e in G **do**
23:     **if** $C_{ex} \cap e \neq \emptyset$ **then**
24:         **if** $G.\ell_E(e) \notin cd.deviceID$ **then**
25:             Add ($G.\ell_E(e)$, $CLASS5$) to cd;

1. Lines 1–9: Initialization
2. Lines 10–14: Add possible devices to CriticalDeviceList cd (CLASS1 and CLASS2)

Query Registration Algorithm (I)

Algorithm 2 register (Range R, DeploymentGraph G)

1: deviceSet $D_c \leftarrow \emptyset$, $D_{uc} \leftarrow \emptyset$;
2: cellSet $C_c \leftarrow \emptyset$, $C_{uc} \leftarrow \emptyset$, $C_{ex} \leftarrow \emptyset$;
3: objectSet $R_c \leftarrow \emptyset$, $R_{uc} \leftarrow \emptyset$;
4: CriticalDeviceList(deviceID, CLASS) $cd \leftarrow \emptyset$;
5: Generate a new identifier $queryID$ for the query;
6: $D_c \leftarrow$ Devices that are covered by R;
7: $D_{uc} \leftarrow$ Devices that intersect with R;
8: $C_c \leftarrow$ Cells which are covered by R;
9: $C_{uc} \leftarrow$ Cells that intersect with R;
10: **for** each device d in D_c **do**
11:   **if** all the cells in $G.\ell_E^{-1}(d)$ are in C_c **then**
12:     Add (d, $CLASS1$) to cd;
13:   **else if** one of the cells in $G.\ell_E^{-1}(d)$ 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** $(C_c \cup C_{uc}) \cap e \neq \emptyset$ AND $(C_c \cup C_{uc}) \cap e \neq (C_c \cup C_{uc})$ **then**
19:     **if** $G.\ell_E(e) \notin cd.deviceID$ **then**
20:       Add $(G.\ell_E(e), CLASS4)$ to cd;
21:     $C_{ex} \leftarrow C_{ex} \cup e \setminus (C_c \cup C_{uc})$;
22: **for** each edge e in G **do**
23:   **if** $C_{ex} \cap e \neq \emptyset$ **then**
24:     **if** $G.\ell_E(e) \notin cd.deviceID$ **then**
25:       Add $(G.\ell_E(e), CLASS5)$ to cd;

1. Lines 1–9: Initialization
2. Lines 10–14: Add possible devices to CriticalDeviceList cd (CLASS1 and CLASS2)
3. Lines 15–16: Add possible CLASS3 devices

Query Registration Algorithm (I)

Algorithm 2 register (Range R, DeploymentGraph G)

1: deviceSet $D_c \leftarrow \emptyset$, $D_{uc} \leftarrow \emptyset$;
2: cellSet $C_c \leftarrow \emptyset$, $C_{uc} \leftarrow \emptyset$, $C_{ex} \leftarrow \emptyset$;
3: objectSet $R_c \leftarrow \emptyset$, $R_{uc} \leftarrow \emptyset$;
4: CriticalDeviceList(deviceID, CLASS) $cd \leftarrow \emptyset$;
5: Generate a new identifier $queryID$ for the query;
6: $D_c \leftarrow$ Devices that are covered by R;
7: $D_{uc} \leftarrow$ Devices that intersect with R;
8: $C_c \leftarrow$ Cells which are covered by R;
9: $C_{uc} \leftarrow$ Cells that intersect with R;
10: **for each** device d in D_c **do**
11:     **if** all the cells in $G.\ell_E^{-1}(d)$ are in C_c **then**
12:         Add $(d, \mathit{CLASS1})$ to cd;
13:     **else if** one of the cells in $G.\ell_E^{-1}(d)$ is in C_{uc} **then**
14:         Add $(d, \mathit{CLASS2})$ to cd;
15: **for each** device d in D_{uc} **do**
16:     Add $(d, \mathit{CLASS3})$ to cd;
17: **for each** edge e in G **do**
18:     **if** $(C_c \cup C_{uc}) \cap e \neq \emptyset$ AND $(C_c \cup C_{uc}) \cap e \neq (C_c \cup C_{uc})$ **then**
19:         **if** $G.\ell_E(e) \notin cd.deviceID$ **then**
20:             Add $(G.\ell_E(e), \mathit{CLASS4})$ to cd;
21:         $C_{ex} \leftarrow C_{ex} \cup e \setminus (C_c \cup C_{uc})$;
22: **for each** edge e in G **do**
23:     **if** $C_{ex} \cap e \neq \emptyset$ **then**
24:         **if** $G.\ell_E(e) \notin cd.deviceID$ **then**
25:             Add $(G.\ell_E(e), \mathit{CLASS5})$ to cd;

1. Lines 1–9: Initialization
2. Lines 10–14: Add possible devices to CriticalDeviceList cd (CLASS1 and CLASS2)
3. Lines 15–16: Add possible CLASS3 devices
4. Lines 17–20: Add possible CLASS4 devices

Query Registration Algorithm (I)

Algorithm 2 register (Range R, DeploymentGraph G)

1: deviceSet $D_c \leftarrow \emptyset$, $D_{uc} \leftarrow \emptyset$;
2: cellSet $C_c \leftarrow \emptyset$, $C_{uc} \leftarrow \emptyset$, $C_{ex} \leftarrow \emptyset$;
3: objectSet $R_c \leftarrow \emptyset$, $R_{uc} \leftarrow \emptyset$;
4: CriticalDeviceList(deviceID, CLASS) $cd \leftarrow \emptyset$;
5: Generate a new identifier $queryID$ for the query;
6: $D_c \leftarrow$ Devices that are covered by R;
7: $D_{uc} \leftarrow$ Devices that intersect with R;
8: $C_c \leftarrow$ Cells which are covered by R;
9: $C_{uc} \leftarrow$ Cells that intersect with R;
10: **for each** device d in D_c **do**
11:     **if** all the cells in $G.\ell_E^{-1}(d)$ are in C_c **then**
12:         Add $(d, CLASS1)$ to cd;
13:     **else if** one of the cells in $G.\ell_E^{-1}(d)$ 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** $(C_c \cup C_{uc}) \cap e \neq \emptyset$ AND $(C_c \cup C_{uc}) \cap e \neq (C_c \cup C_{uc})$ **then**
19:         **if** $G.\ell_E(e) \notin cd.deviceID$ **then**
20:             Add $(G.\ell_E(e), CLASS4)$ to cd;
21:         $C_{ex} \leftarrow C_{ex} \cup e \setminus (C_c \cup C_{uc})$;
22: **for each** edge e in G **do**
23:     **if** $C_{ex} \cap e \neq \emptyset$ **then**
24:         **if** $G.\ell_E(e) \notin cd.deviceID$ **then**
25:             Add $(G.\ell_E(e), CLASS5)$ to cd;

1. Lines 1–9: Initialization
2. Lines 10–14: Add possible devices to CriticalDeviceList cd (CLASS1 and CLASS2)
3. Lines 15–16: Add possible CLASS3 devices
4. Lines 17–20: Add possible CLASS4 devices
5. Line 21: Determine extended cell set C_{ex}

Query Registration Algorithm (I)

Algorithm 2 register (Range R, DeploymentGraph G)

1: deviceSet $D_c \leftarrow \emptyset$, $D_{uc} \leftarrow \emptyset$;
2: cellSet $C_c \leftarrow \emptyset$, $C_{uc} \leftarrow \emptyset$, $C_{ex} \leftarrow \emptyset$;
3: objectSet $R_c \leftarrow \emptyset$, $R_{uc} \leftarrow \emptyset$;
4: CriticalDeviceList(deviceID, CLASS) $cd \leftarrow \emptyset$;
5: Generate a new identifier $queryID$ for the query;
6: $D_c \leftarrow$ Devices that are covered by R;
7: $D_{uc} \leftarrow$ Devices that intersect with R;
8: $C_c \leftarrow$ Cells which are covered by R;
9: $C_{uc} \leftarrow$ Cells that intersect with R;
10: **for each** device d in D_c **do**
11:     **if** all the cells in $G.\ell_E^{-1}(d)$ are in C_c **then**
12:         Add (d, $CLASS1$) to cd;
13:     **else if** one of the cells in $G.\ell_E^{-1}(d)$ 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** $(C_c \cup C_{uc}) \cap e \neq \emptyset$ AND $(C_c \cup C_{uc}) \cap e \neq (C_c \cup C_{uc})$ **then**
19:         **if** $G.\ell_E(e) \notin cd.deviceID$ **then**
20:             Add ($G.\ell_E(e)$, $CLASS4$) to cd;
21:         $C_{ex} \leftarrow C_{ex} \cup e \setminus (C_c \cup C_{uc})$;
22: **for each** edge e in G **do**
23:     **if** $C_{ex} \cap e \neq \emptyset$ **then**
24:         **if** $G.\ell_E(e) \notin cd.deviceID$ **then**
25:             Add ($G.\ell_E(e)$, $CLASS5$) to cd;

1. Lines 1–9: Initialization
2. Lines 10–14: Add possible devices to CriticalDeviceList cd (CLASS1 and CLASS2)
3. Lines 15–16: Add possible CLASS3 devices
4. Lines 17–20: Add possible CLASS4 devices
5. Line 21: Determine extended cell set C_{ex}
6. Lines 22–25: Add possible CLASS5 devices

Query Registration Algorithm (II)

1. **Lines 26–27**: Add active objects from DHT to the certain result

26: **for** each device d in D_c **do**
27: $\quad R_c \leftarrow R_c \cup DHT[d]$;
28: **for** each device d in D_{uc} **do**
29: $\quad R_{uc} \leftarrow R_{uc} \cup DHT[d]$;
30: **for** each cell c in C_c **do**
31: $\quad R_c \leftarrow R_c \cup CDHT[c]$;
32: **if** $|C_c| > 1$ **then**
33: $\quad$ **for** each nondeterministic object o in C_c **do**
34: $\quad\quad$ **if** $OHT[o].IDSet \subset C_c$ **then**
35: $\quad\quad\quad$ Add o into R_c;
36: $\quad\quad$ **else**
37: $\quad\quad\quad$ Add o into R_{uc}
38: **else**
39: $\quad R_{uc} \leftarrow R_{uc} \cup CNHT[c]$;
40: **for** each cell c in C_{uc} **do**
41: $\quad R_c \leftarrow R_c \cup CDHT[c]$; $R_{uc} \leftarrow R_{uc} \cup CNHT[c]$;
42: $QHT[queryID] \leftarrow (R_c, R_{uc})$;
43: **for** each item a in cd **do**
44: $\quad$ Add $(queryID, a.CLASS)$ into $DQHT[a.deviceID]$;

Query Registration Algorithm (II)

26: **for** each device d in D_c **do**
27: $\quad R_c \leftarrow R_c \cup DHT[d]$;
28: **for** each device d in D_{uc} **do**
29: $\quad R_{uc} \leftarrow R_{uc} \cup DHT[d]$;
30: **for** each cell c in C_c **do**
31: $\quad R_c \leftarrow R_c \cup CDHT[c]$;
32: **if** $|C_c| > 1$ **then**
33: $\quad$ **for** each nondeterministic object o in C_c **do**
34: $\quad\quad$ **if** $OHT[o].IDSet \subset C_c$ **then**
35: $\quad\quad\quad$ Add o into R_c;
36: $\quad\quad$ **else**
37: $\quad\quad\quad$ Add o into R_{uc}
38: **else**
39: $\quad R_{uc} \leftarrow R_{uc} \cup CNHT[c]$;
40: **for** each cell c in C_{uc} **do**
41: $\quad R_c \leftarrow R_c \cup CDHT[c]$; $R_{uc} \leftarrow R_{uc} \cup CNHT[c]$;
42: $QHT[queryID] \leftarrow (R_c, R_{uc})$;
43: **for** each item a in cd **do**
44: $\quad$ Add $(queryID, a.CLASS)$ into $DQHT[a.deviceID]$;

1. **Lines 26–27**: Add active objects from DHT to the certain result
2. **Lines 28–29**: Intersected device set, add active objects from DHT to the uncertain result

Query Registration Algorithm (II)

26: **for** each device d in D_c **do**
27: $\quad R_c \leftarrow R_c \cup DHT[d]$;
28: **for** each device d in D_{uc} **do**
29: $\quad R_{uc} \leftarrow R_{uc} \cup DHT[d]$;
30: **for** each cell c in C_c **do**
31: $\quad R_c \leftarrow R_c \cup CDHT[c]$;
32: **if** $|C_c| > 1$ **then**
33: $\quad$ **for** each nondeterministic object o in C_c **do**
34: $\quad\quad$ **if** $OHT[o].IDSet \subset C_c$ **then**
35: $\quad\quad\quad$ Add o into R_c;
36: $\quad\quad$ **else**
37: $\quad\quad\quad$ Add o into R_{uc}
38: **else**
39: $\quad R_{uc} \leftarrow R_{uc} \cup CNHT[c]$;
40: **for** each cell c in C_{uc} **do**
41: $\quad R_c \leftarrow R_c \cup CDHT[c]$; $R_{uc} \leftarrow R_{uc} \cup CNHT[c]$;
42: $QHT[queryID] \leftarrow (R_c, R_{uc})$;
43: **for** each item a in cd **do**
44: $\quad$ Add $(queryID, a.CLASS)$ into $DQHT[a.deviceID]$;

1. Lines 26–27: Add active objects from DHT to the certain result
2. Lines 28–29: Intersected device set, add active objects from DHT to the uncertain result
3. Lines 30–31: From covered cells, add deterministic objects to the certain result

Query Registration Algorithm (II)

26: **for** each device d in D_c **do**
27: $\quad R_c \leftarrow R_c \cup DHT[d]$;
28: **for** each device d in D_{uc} **do**
29: $\quad R_{uc} \leftarrow R_{uc} \cup DHT[d]$;
30: **for** each cell c in C_c **do**
31: $\quad R_c \leftarrow R_c \cup CDHT[c]$;
32: **if** $|C_c| > 1$ **then**
33: $\quad$ **for** each nondeterministic object o in C_c **do**
34: $\quad\quad$ **if** $OHT[o].IDSet \subset C_c$ **then**
35: $\quad\quad\quad$ Add o into R_c;
36: $\quad\quad$ **else**
37: $\quad\quad\quad$ Add o into R_{uc}
38: **else**
39: $\quad R_{uc} \leftarrow R_{uc} \cup CNHT[c]$;
40: **for** each cell c in C_{uc} **do**
41: $\quad R_c \leftarrow R_c \cup CDHT[c]$; $R_{uc} \leftarrow R_{uc} \cup CNHT[c]$;
42: $QHT[queryID] \leftarrow (R_c, R_{uc})$;
43: **for** each item a in cd **do**
44: $\quad$ Add $(queryID, a.CLASS)$ into $DQHT[a.deviceID]$;

1. Lines 26–27: Add active objects from DHT to the certain result
2. Lines 28–29: Intersected device set, add active objects from DHT to the uncertain result
3. Lines 30–31: From covered cells, add deterministic objects to the certain result
4. Lines 32-37: If more than one cell, check nondeterministic objects. If all its possible cells are in C_c add the object to the certain result, else uncertain result

Query Registration Algorithm (II)

26: **for** each device d in D_c **do**
27: $\quad R_c \leftarrow R_c \cup DHT[d]$;
28: **for** each device d in D_{uc} **do**
29: $\quad R_{uc} \leftarrow R_{uc} \cup DHT[d]$;
30: **for** each cell c in C_c **do**
31: $\quad R_c \leftarrow R_c \cup CDHT[c]$;
32: **if** $|C_c| > 1$ **then**
33: $\quad$ **for** each nondeterministic object o in C_c **do**
34: $\quad\quad$ **if** $OHT[o].IDSet \subset C_c$ **then**
35: $\quad\quad\quad$ Add o into R_c;
36: $\quad\quad$ **else**
37: $\quad\quad\quad$ Add o into R_{uc}
38: **else**
39: $\quad R_{uc} \leftarrow R_{uc} \cup CNHT[c]$;
40: **for** each cell c in C_{uc} **do**
41: $\quad R_c \leftarrow R_c \cup CDHT[c]$; $R_{uc} \leftarrow R_{uc} \cup CNHT[c]$;
42: $QHT[queryID] \leftarrow (R_c, R_{uc})$;
43: **for** each item a in cd **do**
44: $\quad$ Add $(queryID, a.CLASS)$ into $DQHT[a.deviceID]$;

1. Lines 26–27: Add active objects from DHT to the certain result
2. Lines 28–29: Intersected device set, add active objects from DHT to the uncertain result
3. Lines 30–31: From covered cells, add deterministic objects to the certain result
4. Lines 32-37: If more than one cell, check nondeterministic objects. If all its possible cells are in C_c add the object to the certain result, else uncertain result
5. Lines 38–39: Only one cell. Nondeterministic objects are added to the uncertain result

Query Registration Algorithm (II)

26: **for** each device d in D_c **do**
27: $\quad R_c \leftarrow R_c \cup DHT[d]$;
28: **for** each device d in D_{uc} **do**
29: $\quad R_{uc} \leftarrow R_{uc} \cup DHT[d]$;
30: **for** each cell c in C_c **do**
31: $\quad R_c \leftarrow R_c \cup CDHT[c]$;
32: **if** $|C_c| > 1$ **then**
33: $\quad$ **for** each nondeterministic object o in C_c **do**
34: $\quad\quad$ **if** $OHT[o].IDSet \subset C_c$ **then**
35: $\quad\quad\quad$ Add o into R_c;
36: $\quad\quad$ **else**
37: $\quad\quad\quad$ Add o into R_{uc}
38: **else**
39: $\quad R_{uc} \leftarrow R_{uc} \cup CNHT[c]$;
40: **for** each cell c in C_{uc} **do**
41: $\quad R_c \leftarrow R_c \cup CDHT[c]$; $\; R_{uc} \leftarrow R_{uc} \cup CNHT[c]$;
42: $QHT[queryID] \leftarrow (R_c, R_{uc})$;
43: **for** each item a in cd **do**
44: $\quad$ Add $(queryID, a.CLASS)$ into $DQHT[a.deviceID]$;

1. Lines 26–27: Add active objects from DHT to the certain result
2. Lines 28–29: Intersected device set, add active objects from DHT to the uncertain result
3. Lines 30–31: From covered cells, add deterministic objects to the certain result
4. Lines 32-37: If more than one cell, check nondeterministic objects. If all its possible cells are in C_c add the object to the certain result, else uncertain result
5. Lines 38–39: Only one cell. Nondeterministic objects are added to the uncertain result
6. Lines 40–41: Intersected set. Add all objects to the uncertain result

Query Registration Algorithm (II)

26: **for** each device d in D_c **do**
27: $\quad R_c \leftarrow R_c \cup DHT[d]$;
28: **for** each device d in D_{uc} **do**
29: $\quad R_{uc} \leftarrow R_{uc} \cup DHT[d]$;
30: **for** each cell c in C_c **do**
31: $\quad R_c \leftarrow R_c \cup CDHT[c]$;
32: **if** $|C_c| > 1$ **then**
33: $\quad$ **for** each nondeterministic object o in C_c **do**
34: $\quad\quad$ **if** $OHT[o].IDSet \subset C_c$ **then**
35: $\quad\quad\quad$ Add o into R_c;
36: $\quad\quad$ **else**
37: $\quad\quad\quad$ Add o into R_{uc}
38: **else**
39: $\quad R_{uc} \leftarrow R_{uc} \cup CNHT[c]$;
40: **for** each cell c in C_{uc} **do**
41: $\quad R_c \leftarrow R_c \cup CDHT[c]$; $R_{uc} \leftarrow R_{uc} \cup CNHT[c]$;
42: $QHT[queryID] \leftarrow (R_c, R_{uc})$;
43: **for** each item a in cd **do**
44: $\quad$ Add $(queryID, a.CLASS)$ into $DQHT[a.deviceID]$;

1. Lines 26–27: Add active objects from DHT to the certain result
2. Lines 28–29: Intersected device set, add active objects from DHT to the uncertain result
3. Lines 30–31: From covered cells, add deterministic objects to the certain result
4. Lines 32-37: If more than one cell, check nondeterministic objects. If all its possible cells are in C_c add the object to the certain result, else uncertain result
5. Lines 38–39: Only one cell. Nondeterministic objects are added to the uncertain result
6. Lines 40–41: Intersected set. Add all objects to the uncertain result
7. Line 42: Results added to QHT

Query Registration Algorithm (II)

26: **for** each device d in D_c **do**
27: $\quad R_c \leftarrow R_c \cup DHT[d]$;
28: **for** each device d in D_{uc} **do**
29: $\quad R_{uc} \leftarrow R_{uc} \cup DHT[d]$;
30: **for** each cell c in C_c **do**
31: $\quad R_c \leftarrow R_c \cup CDHT[c]$;
32: **if** $|C_c| > 1$ **then**
33: $\quad$ **for** each nondeterministic object o in C_c **do**
34: $\quad\quad$ **if** $OHT[o].IDSet \subset C_c$ **then**
35: $\quad\quad\quad$ Add o into R_c;
36: $\quad\quad$ **else**
37: $\quad\quad\quad$ Add o into R_{uc}
38: **else**
39: $\quad R_{uc} \leftarrow R_{uc} \cup CNHT[c]$;
40: **for** each cell c in C_{uc} **do**
41: $\quad R_c \leftarrow R_c \cup CDHT[c]$; $R_{uc} \leftarrow R_{uc} \cup CNHT[c]$;
42: $QHT[queryID] \leftarrow (R_c, R_{uc})$;
43: **for** each item a in cd **do**
44: $\quad$ Add $(queryID, a.CLASS)$ into $DQHT[a.deviceID]$;

1. Lines 26–27: Add active objects from DHT to the certain result
2. Lines 28–29: Intersected device set, add active objects from DHT to the uncertain result
3. Lines 30–31: From covered cells, add deterministic objects to the certain result
4. Lines 32-37: If more than one cell, check nondeterministic objects. If all its possible cells are in C_c add the object to the certain result, else uncertain result
5. Lines 38–39: Only one cell. Nondeterministic objects are added to the uncertain result
6. Lines 40–41: Intersected set. Add all objects to the uncertain result
7. Line 42: Results added to QHT
8. Lines 43–44: DQHT entry is created for each critical device

Query Result Updates

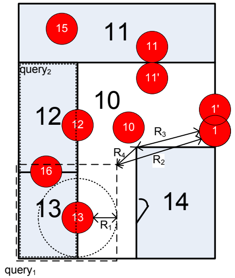
- When an object enters the activation range of a critical device:
  - For $\mathrm{CLASS1}$ or $\mathrm{CLASS2}$ devices the object is the certain result
  - For $\mathrm{CLASS3}$ devices the object is possibly in the query range
  - For $\mathrm{CLASS4}$ or $\mathrm{CLASS5}$ devices the object is not in the query range
- When an object leaves:
  - For $\mathrm{CLASS1}, 3, 5$ devices there are no changes
  - For $\mathrm{CLASS2}$ devices the object may still be in the query range, thus it is moved to the uncertain result
  - For $\mathrm{CLASS4}$ devices the object may be in a cell that intersects with the query range and it added to the uncertain result

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

Deferred Query Updates

- Deferred query updates is the concept of postponing updates where we already know the result
- The time a query result is still valid is calculated from *minimum indoor walking distance* divided by the *maximum speed* an object can travel

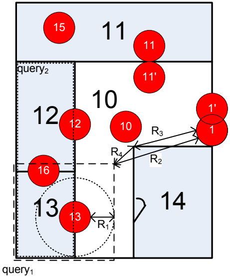


Consider $query_1$, after the object o leaves a $\mathrm{CLASS2}$ critical device $devoce_{13}$, it should be moved from certain to uncertain result.

Let V_{max} be the maximum speed, if $R_1 = V_{max} * \Delta t$, the certain result can be maintained without updating for an extra period of time Δt.

Deferred Query Updates

- Deferred query updates is the concept of postponing updates where we already know the result
- The time a query result is still valid is calculated from *minimum indoor walking distance* divided by the *maximum speed* an object can travel



Consider $query_1$, after the object o leaves a CLASS2 critical device $devoce_{13}$, it should be moved from certain to uncertain result.

Let V_{max} be the maximum speed, if $R_1 = V_{max} * \Delta t$, the certain result can be maintained without updating for an extra period of time Δt.

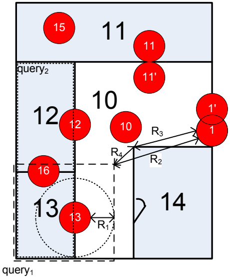
Probabilistic Analysis of Uncertain Results

To analyze probability that o is in the query range R. Assume that the possible locations in a given indoor space conform a uniform distribution within all reachable regions constrained by its maximum speed.

1. Probabilities for Active Objects

Formally, the probability that an active object o is in the range R is defined as:

$$prob(o\Theta R) = \frac{Area(Devices(d).ActRange \sqcap R)}{Area(Devices(d).ActRange)} \quad (1)$$



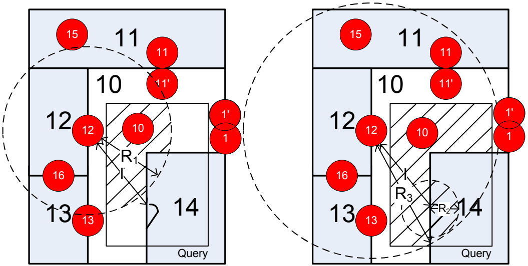
Consider $device_{16}$, a CLASS3 device for $query_1$, the probability for an active object in $device_{16}$ to be in the query range is calculated as ...

Probabilistic Analysis of Uncertain Results

2. Probabilities for Inactive Objects

For the case that after leaving $\mathrm{CLASS}2, 3, 4$ devices, the probabilities for inactive objects can be defined based on the maximum speed constraint.

An example for an inactive object that just leaves $device_{12}$...



Conclusion

- A solution with a symbolic model of the floor plan, device locations, and activation ranges
- Data is stored in several hash tables which make it possible to efficiently locate a specific object (result is a signle room/cell, or a set of rooms/cells)
- Future work
  - sharing of query processing among concurrent queries
  - common critical devices exploitation
  - other types of queries: range and kNN
  - further investigate the probability analysis

1. Outlines
2. Indoor Space Models & Applications
3. Indoor Data Cleansing
4. Indoor Movement Analysis
5. Appendix

References

M. F. Worboys.
Modeling indoor space.
In *ISA*, pp. 1–6, 2011.

X. Xie, H. Lu, and T. B. Pedersen.
Efficient distance-aware query evaluation on indoor moving objects.
In *ICDE*, pp. 434–445, 2013.

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

B. Yang, H. Lu, and C. S. Jensen.
Probabilistic threshold k nearest neighbor queries over moving objects in symbolic indoor space.
In *EDBT*, pp. 335–346, 2010.

References

H. Lu, X. Cao, and C. S. Jensen.
A foundation for efficient indoor distance-aware query processing.
In *ICDE*, pp. 438–449, 2012.

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

H. Lu, B. Yang, and C. S. Jensen.
Spatio-temporal Joins on Symbolic Indoor Tracking Data.
In *ICDE*, pp. 816–827, 2011.

The End. Thanks :)