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

Huan Li

Database Laboratory, Zhejiang University

lihuancs@zju.edu.cn

March 24, 2016

Overview

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

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

1. Outlines

2. Indoor Space Models & Applications

3. Indoor Data Cleansing

4. Indoor Movement Analysis

5. Appendix

About This Work...

Probabilistic Threshold k Nearest Neighbor Queries over Moving Objects in Symbolic Indoor Space. [3]
B. Yang, H. Lu, and C. S. Jensen.

- Published in year 2010 at the *EDBT* conference.
- **Minimal Indoor Walking Distance**(MIWD) along with algorithms and data structures are proposed for distance computing and storage.
- Effective object indexing structures, also capture the uncertainty of object locations.
- On this foundation, Probabilistic threshold kNN (PTkNN) query is studied.

Motivation

- Indoor positioning makes it possible to support interesting queries over large populations of moving objects.[4]
  - shopping mall, airports, office buildings
  - kNN queries over indoor moving objects enables the detection of approaching potential threats at sensitive locations in a subway system
- Existing kNN techniques in spatial and spatiotemporal databases are inapplicable in indoor spaces.
  - complex entities and topologies
  - indoor positioning techniques differ fundamentally from outdoor GPS, low sampling frequency and accuracy

Minimal Indoor Walking Distance

Minimal Indoor Walking Distance(MIWD) is used as the distance metric in indoor spaces.

Minimal Indoor Walking Distance

Minimal Indoor Walking Distance(MIWD) is used as the distance metric in indoor spaces.

The mapping $Rooms$ determines the room of an indoor position:

Minimal Indoor Walking Distance

Minimal Indoor Walking Distance(MIWD) is used as the distance metric in indoor spaces.
The mapping $Rooms$ determines the room of an indoor position:

$$Rooms : positions \rightarrow \Sigma_{rooms} \tag{1}$$

Minimal Indoor Walking Distance

Minimal Indoor Walking Distance(MIWD) is used as the distance metric in indoor spaces.

The mapping $Rooms$ determines the room of an indoor position:

$$Rooms : positions \rightarrow \Sigma_{rooms} \tag{1}$$

The mapping $Doors$ maps a room to the doors that connect the room to an adjacent room:

Minimal Indoor Walking Distance

Minimal Indoor Walking Distance(MIWD) is used as the distance metric in indoor spaces.

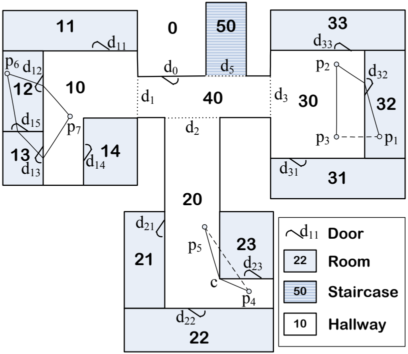
The mapping $Rooms$ determines the room of an indoor position:

$$Rooms : positions \to \Sigma_{rooms} \tag{1}$$

The mapping $Doors$ maps a room to the doors that connect the room to an adjacent room:

$$Doors : \Sigma_{rooms} \to 2^{\Sigma_{doors}} \tag{2}$$

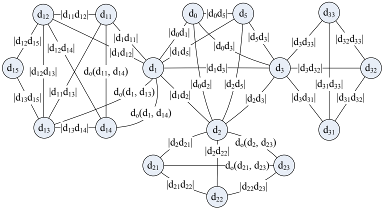
Minimal Indoor Walking Distance



- intra-room obstructed distance, termed as d_o. E.g., $d_o(p_2, p_3) = |p_2 p_3|$ and $d_o(p_4, p_5) = |p_4 c| + |c p_5|$.
- if in different rooms, it should take into account the doors connecting the rooms. E.g., $d_{MIN}(p_1, p_2) = |p_1 d_{32}| + |d_{17} p_9|$.
- if there exist several paths, the correct path should be the shortest one. E.g., $d_{MIN}(p_6, p_7) = |p_6 d_{12}| + |d_{12} p_7| \neq |p_6 d_{15}| + |d_{15} d_{13}| + |d_{13} p_7|$.

Minimal Indoor Walking Distance

Doors Graph is capable of retrieving the connecting doors between two rooms, which is convenient for computing MIWD.



Doors Graph

- $G_d = \{D, E, l_{weight}\}$
- $D = \Sigma_{doors}$ is the set of the vertices
- E: An edge $\{d_i, d_j\}$ exists if a room rm exists in Σ_{rooms} such that $\{d_i, d_j\} \subseteq Doors(rm)$
- $l_{weight} : E \rightarrow R$ assigns to an edge the obstructed distance between the two doors as $d_o(d_i, d_j)$

Minimal Indoor Walking Distance

All **door-to-door shortest path distances** can be computed and recorded in a hash table $\mathrm{D2D}$ according to the Doors Graph.

Minimal Indoor Walking Distance

All **door-to-door shortest path distances** can be computed and recorded in a hash table $\mathrm{D2D}$ according to the Doors Graph.

$$\mathrm{D2D} : \{(d_p, d_j)\} \to R, \quad d_p, d_q \in \Sigma_{doors}. \tag{3}$$

Minimal Indoor Walking Distance

All **door-to-door shortest path distances** can be computed and recorded in a hash table $\mathrm{D2D}$ according to the Doors Graph.

$$\mathrm{D2D} : \{(d_p, d_j)\} \to R, \quad d_p, d_q \in \Sigma_{doors}. \tag{3}$$

Consequently, for two positions p and q in different rooms.

Minimal Indoor Walking Distance

All **door-to-door shortest path distances** can be computed and recorded in a hash table $\mathrm{D2D}$ according to the Doors Graph.

$$\mathrm{D2D} : \{(d_p, d_j)\} \rightarrow R, \quad d_p, d_q \in \Sigma_{doors}. \tag{3}$$

Consequently, for two positions p and q in different rooms.

$$d_{MIN}(d_p, d_q) = d_o(p, d_p) + \mathrm{D2D}(d_p, d_q) + d_o(d_q, q) \tag{4}$$

where $d_p(d_q)$ ranges over all doors of room $p(q)$.

Minimal Indoor Walking Distance

All **door-to-door shortest path distances** can be computed and recorded in a hash table $\mathrm{D2D}$ according to the Doors Graph.

$$\mathrm{D2D} : \{(d_p, d_j)\} \to R, \quad d_p, d_q \in \Sigma_{doors}. \tag{3}$$

Consequently, for two positions p and q in different rooms.

$$d_{MIN}(d_p, d_q) = d_o(p, d_p) + \mathrm{D2D}(d_p, d_q) + d_o(d_q, q) \tag{4}$$

where $d_p(d_q)$ ranges over all doors of room $p(q)$.

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

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

it is possible to adapt this notion of distance to accommodate other semantics. For example, a person might prefer a longer indoor path that passes as few doors as possible.

Symbolic Indoor Positioning

each device detects and reports the observed objects at a relatively high sampling rate.

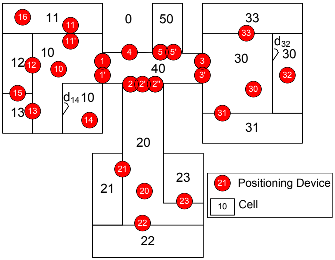
A reading $(deviceID, objectID, t)$ states that object $objectID$ is detected by device $objectID$ at time t.

Only its first and last appearances in a device's range are of interest.

- $flag = ENTER$ indicates that the object is entering the device's activation range
- $flag = LEAVE$ indicates that the object is leaving the device's activation range

Positioning Devices Deployment Graph [2]

- 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*

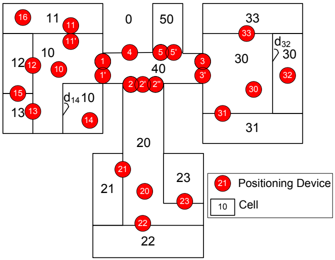


Deployment Graph

- $G = \{C, E, \Sigma_{devices}, l_E\}$
- 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., $E \rightarrow 2^{\Sigma_{devices}}$

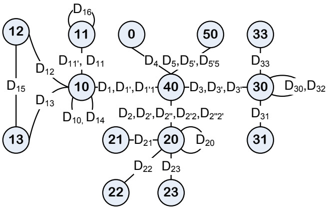
Positioning Devices Deployment Graph [2]

- 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*

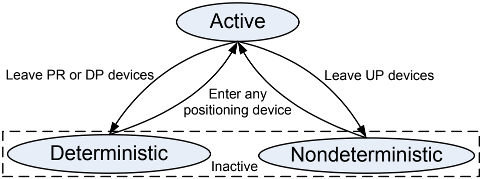


Deployment Graph

- $G = \{C, E, \Sigma_{devices}, l_E\}$
- 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 [2]



- 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 [2]

The proposed indexing scheme uses 4 hash tables

Indexing Indoor Moving Objects [2]

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 [2]

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 [2]

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 [2]

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 [2]

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 [2]

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 [2]

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 [2]

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 [2]

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 [2]

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

IDSet $sSet \leftarrow \emptyset$;
if $O.flag$ = $ENTER$ **then**
    $sSet \leftarrow OHT[O.objectID].IDSet$;
    if $OHT[O.objectID].STATE$ = $Active$ **then**
        for the single element c in $sSet$ **do**
            Delete $O.objectID$ from $DHT[c]$;
    else if $OHT[O.objectID].STATE$ = $Deterministic$ **then**
        for the single element c in $sSet$ **do**
            Delete $O.objectID$ from $CDHT[c]$;
    else
        for each element c in $sSet$ **do**
            Delete $O.objectID$ from $CNHT[c]$;
    Add $O.objectID$ to $DHT[O.deviceID]$;
    $OHT[O.objectID] \leftarrow (Active, O.t, \{O.deviceID\})$;
else
    Delete $O.objectID$ from $DHT[O.deviceID]$;
    $sSet \leftarrow G.\ell_E^{-1}(O.deviceID)$;
    if $Devices(O.deviceID).TYPE$ = UP **then**
        $OHT[O.objectID] \leftarrow (Nondeterministic, O.t, sSet)$;
        for each element c in $sSet$ **do**
            Add $O.objectID$ to $CNHT[c]$;
    else
        $OHT[O.objectID] \leftarrow (Deterministic, O.t, sSet)$;
        for the single element c in $sSet$ **do**
            Add $O.objectID$ to $CDHT[c]$;

RFID Deployment Graph Construction [2]

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 [2]

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

RFID Deployment Graph Construction [2]

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
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 [2]

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

RFID Deployment Graph Construction [2]

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

Deriving Uncertain Regions for Indoor Moving Objects

For outdoor moving objects [5], **Uncertainty Region**, denoted by $UR(o,t)$, is a region such that o must be in this region at time t.

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

$$\int_{UR(o_i,t)} f_{o_i}(x,y,t)dxdy = 1 \quad (5)$$

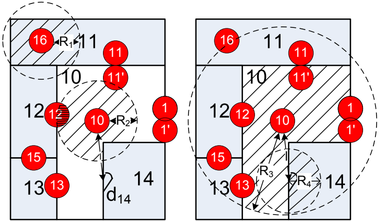
Deriving Uncertain Regions for Indoor Moving Objects

Assume that an *indoor object* has the same probability to be located anywhere inside its uncertainty region:

$$f_{o_i}(x, y, t) = \frac{1}{Area(UR(o_i, t))}, \quad (x, y) \in UR(o_i, t) \qquad (6)$$

- **Active object**: the activation range of the corresponding device.
- **Deterministic object**: the intersection between the object's cell and its *maximum-speed constrained circle*.
- **Nondeterministic object**: the union of the intersection between each cell and the circle.

Maximum-speed Constrained Circle



- assume o left $device_{16}$ at time t, intersection of maximum-speed constrained circle $C_{MSC}(o, device_{16}, t)$ and cell c_{11}.
- suppose o left $device_{10}$ at time t, UR is the intersection between room 10 and $C_{MSC}(o, device_{10}, t)$.

- as time t passes by, UR therefore contains two parts...
- uncertainty region of an active object can be refined as the intersection of the activation range of device and C_{MSC}. suppose o left $device_{10}$ and then is detected by $device_{12}$, its uncertainty region is the intersection of $C_{MSC}(o, device_{10}, t)$ and activation range of $device_{12}$.

PTkNN Query Processing

Definition (Indoor Probabilistic Threshold kNN Query)

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

when the number of moving objects increases, the number of k-subsets (A in Definition 1) in the result set $\mathbb{R}$ increase exponentially. Specifically, there are $\binom{n}{k}$ possible k-subsets for a PTkNN query over n objects.

PTkNN Query Processing

Three techniques are proposed to speed up PTkNN query processing:

1. minimum indoor walking distances between the query location and the (uncertainty regions of) objects are used to prune the objects too far away to be in any possible k-subset, results in a subset $O' \subseteq O$.
2. for all k-subsets of O', cost-efficient probability estimates are used to prune the k-subsets whose probabilities definitely are lower than the threshold T.
3. for each remaining k-subset A, add A to $\mathbb{R}$ only if $prob(A) > T$.

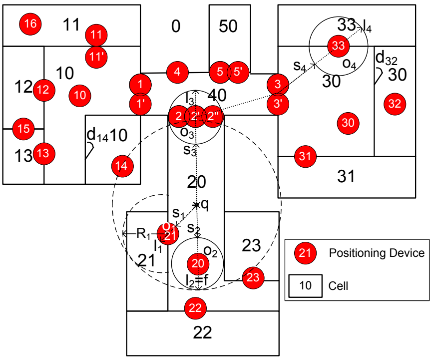
PTkNN Query Processing

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

PTkNN Query Processing

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

$$s_i = \min_{p \in UR(o_i,t)} d_{MIW}(q,p); \quad l_i = \max_{p \in UR(o_i,t)} d_{MIW}(q,p) \qquad (7)$$



- o_1 is being observed by $device_{12}$
- o_2, o_3, o_4 recently left $device_{20}, device_{2'}, device_{33}$ respectively
- o_4's MIWD computation should take into account the door connectivity

PTkNN Query Processing

Definition (k-bound)

Let f be the kth minimal one of all objects' l_is. If s_i of object o_i is greater than f, object o_i has no chance to be in any k-subset of the result set $\mathbb{R}$. This f value is called the k-bound. [6]

- k-bound can be calculated and updated dynamically during the distance based pruning as soon as k objects have been seen from O.
- by taking advantage of indoor space distance definition, the computation cost can be further reduced. E.g., the objects in the same cell can be pruned if one is pruned by the k-bound.

Distance Pruning

Algorithm 4 DistancePruning(Position q, int k)

ObjectSet $O' \leftarrow \emptyset$;
Double $f \leftarrow +\infty$;
CellSet $seeds \leftarrow \emptyset$;
Initialize a min-heap $H\langle\langle d, v\rangle\rangle$;
if q is in the activation range of a device dev **then**
    $O' \leftarrow DHT[dev]$; $seeds \leftarrow G.\ell_E^{-1}(dev)$;
    for each cell c in $seeds$ **do**
        $O' \leftarrow O' \cup CDHT[c] \cup CNHT[c]$; $EnheapDoors(H, c)$;
else
    Room $r \leftarrow Rooms(q)$;
    Cell $c \leftarrow Cells^{-1}(r)$;
    $O' \leftarrow CDHT[c] \cup CNHT[c]$;
    Add c into $seeds$; $EnheapDoors(H, c)$;
if $|O'| \geq k$ **then**
    $f \leftarrow Bound(O')$;
while H is not empty **do**
    $e \leftarrow deheap(H)$;
    if $e.v > f$ **then**
        break;
    Set $e.d$ as visited;
    $dev \leftarrow PA2D^{-1}(e.d)$; $O' \leftarrow O' \cup DHT[dev]$;
    for each cell c in $G.\ell_E^{-1}(dev)$ **do**
        if $c \notin seeds$ **then**
            $O' \leftarrow O' \cup CDHT[c] \cup CNHT[c]$;
            for each dev in $G.\ell_E(\{c, c\})$ **do**
                if $(PR2D(dev, d) + e.v) \leq f$ **then**
                    $O' \leftarrow O' \cup DHT[dev]$;
            Add c into $seeds$; $EnheapDoors(H, c)$;
    if $|O'| \geq k$ **then**
        $f \leftarrow Bound(O')$;

Distance Pruning

Algorithm 4 DistancePruning(Position q, int k)

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

1. Lines 1–4: initialization: candidate object set O', k-bound f, cell set $seeds$ records the examined cells, min-heap $H\langle\langle d, v\rangle\rangle$ gives priority to doors closer to location q

Distance Pruning

Algorithm 4 DistancePruning(Position q, int k)

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

1. Lines 1–4: initialization: candidate object set O', k-bound f, cell set $seeds$ records the examined cells, min-heap $H\langle\langle d, v\rangle\rangle$ gives priority to doors closer to location q
2. Lines 5–8: if q is in dev's activation range, active objects in dev are added to O', the nearby objects are added are also added to O'

Distance Pruning

Algorithm 4 DistancePruning(Position q, int k)

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

1. Lines 1–4: initialization: candidate object set O', k-bound f, cell set *seeds* records the examined cells, min-heap $H\langle\langle d, v\rangle\rangle$ gives priority to doors closer to location q
2. Lines 5–8: if q is in dev's activation range, active objects in dev are added to O', the nearby objects are added are also added to O'
3. Lines 9–13: if not, the covering cell c is added to set *seeds*, add both deterministic and nondeterministic objects to O'

Distance Pruning

Algorithm 4 DistancePruning(Position q, int k)

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

1. Lines 1–4: initialization: candidate object set O', k-bound f, cell set *seeds* records the examined cells, min-heap $H\langle\langle d, v\rangle\rangle$ gives priority to doors closer to location q
2. Lines 5–8: if q is in dev's activation range, active objects in dev are added to O', the nearby objects are added are also added to O'
3. Lines 9–13: if not, the covering cell c is added to set *seeds*, add both deterministic and nondeterministic objects to O'
4. Lines 14–15: determine the k-bound from the current O'

Distance Pruning

Algorithm 4 DistancePruning(Position q, int k)

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

1. Lines 1–4: initialization: candidate object set O', k-bound f, cell set $seeds$ records the examined cells, min-heap $H\langle\langle d, v\rangle\rangle$ gives priority to doors closer to location q
2. Lines 5–8: if q is in dev's activation range, active objects in dev are added to O', the nearby objects are added are also added to O'
3. Lines 9–13: if not, the covering cell c is added to set $seeds$, add both deterministic and nondeterministic objects to O'
4. Lines 14–15: determine the k-bound from the current O'
5. Lines 17-30: expansion step followed Dijkstra, stops when the entry's distance is too far away, if $|O'| > k$ then update f and reduce the candidate set O'

Distance Pruning

Algorithm 4 DistancePruning(Position q, int k)

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

1. Lines 1–4: initialization: candidate object set O', k-bound f, cell set *seeds* records the examined cells, min-heap $H\langle\langle d, v\rangle\rangle$ gives priority to doors closer to location q
2. Lines 5–8: if q is in dev's activation range, active objects in dev are added to O', the nearby objects are added are also added to O'
3. Lines 9–13: if not, the covering cell c is added to set *seeds*, add both deterministic and nondeterministic objects to O'
4. Lines 14–15: determine the k-bound from the current O'
5. Lines 17-30: expansion step followed Dijkstra, stops when the entry's distance is too far away, if $|O'| > k$ then update f and reduce the candidate set O'

Probability Threshold Based Pruning

There can still be $\binom{|O'|}{k}$ *possible* k*-subsets in result set* $\mathbb{R}$.

Probability Threshold Based Pruning

There can still be $\binom{|O'|}{k}$ possible k-subsets in result set $\mathbb{R}$.

Given an object o_i, let $P_{o_i}(r)$ be the cumulative distribution function that o_i's MIWD to the query location q is r:

Probability Threshold Based Pruning

There can still be $\binom{|O'|}{k}$ *possible* k*-subsets in result set* $\mathbb{R}$.

Given an object o_i, let $P_{o_i}(r)$ be the cumulative distribution function that o_i's MIWD to the query location q is r:

$$P_{o_i}(r) = Pr(d_{MIW}(q, o_i) \leq r) \tag{8}$$

Let A be a k-subset of O'. The probability $Prob(A)$ that A contains the k nearest neighbors of q satisfies:

Probability Threshold Based Pruning

There can still be $\binom{|O'|}{k}$ possible k-subsets in result set $\mathbb{R}$.

Given an object o_i, let $P_{o_i}(r)$ be the cumulative distribution function that o_i's MIWD to the query location q is r:

$$P_{o_i}(r) = Pr(d_{MIW}(q, o_i) \leq r) \quad (8)$$

Let A be a k-subset of O'. The probability $Prob(A)$ that A contains the k nearest neighbors of q satisfies:

$$Prob(A) \leq \prod_{o_i \in A} P_{o_i}(f) \quad (9)$$

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

Probability Threshold Based Pruning

There can still be $\binom{|O'|}{k}$ possible k-subsets in result set $\mathbb{R}$.

Given an object o_i, let $P_{o_i}(r)$ be the cumulative distribution function that o_i's MIWD to the query location q is r:

$$P_{o_i}(r) = Pr(d_{MIW}(q, o_i) \leq r) \tag{8}$$

Let A be a k-subset of O'. The probability $Prob(A)$ that A contains the k nearest neighbors of q satisfies:

$$Prob(A) \leq \prod_{o_i \in A} P_{o_i}(f) \tag{9}$$

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

$$Prob(A) \leq \prod_{o_i \in A} P_{o_i}(f) \leq P_{o_i}(f) \leq T \tag{10}$$

Therefore, if $P_{o_i}(f) \leq T$, o_i can be safely pruned from the candidate.

Probability Evaluation

After probability threshold pruning, each k-subset A in $\mathbb{R}$ may have their $Prob(A)$ greater than T.

Define $Prob(A)$ as:

Probability Evaluation

After probability threshold pruning, each k-subset A in $\mathbb{R}$ may have their $Prob(A)$ greater than T.

Define $Prob(A)$ as:

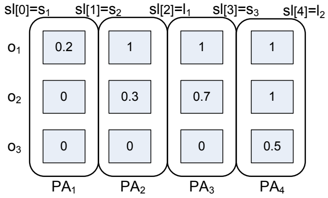
$$Prob(A) = \sum_{o_z \in A} \int_0^{+\infty} p_{o_z}(r) \prod_{o_i \in A \setminus \{o_z\}} P_{o_i}(r) \prod_{o_j \in O' \setminus A} (1 - P_{o_j}(r))dr \tag{11}$$

Probability Evaluation

After probability threshold pruning, each k-subset A in $\mathbb{R}$ may have their $Prob(A)$ greater than T.

Define $Prob(A)$ as:

$$Prob(A) = \sum_{o_z \in A} \int_0^{+\infty} p_{o_z}(r) \prod_{o_i \in A \setminus \{o_z\}} P_{o_i}(r) \prod_{o_j \in O' \setminus A} (1 - P_{o_j}(r)) dr \tag{11}$$



- o_z is the k-th nearest neighbor of q.
- to evaluate the probabilities efficiently, a partition based approximate evaluation method is used.
- let an array sl records all the minimum distances s_i in O' and the maximum distances l_i that satisfy $l_i \leq f$.
- sl should have $g = |O'| + k$ elements, sort it in ascending order.

Research Directions

- Analyzing historical trajectory data may discover associations among object movements, which can be used to design more efficient group pruning in processing a PTkNN query
- Regarding the uncertainty model of indoor moving objects, it is also interesting to conduct probabilistic analysis on other kinds of object distribution.

References I

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

[2] 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.

[3] 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.

[4] C. S. Jensen, H. Lu and B. Yang.
Indoor-A New Data Management Frontier.
In *IEEE Data Eng. Bull.*, pp. 12–17, 2010.

References II

[5] R. Cheng, D. V. Kalashnikov and S.Prabhakar.
Querying imprecise data in moving object environments.
In *TKDE*, pp. 1112–1127, 2004.

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

1 1. Outlines

2 2. Indoor Space Models & Applications

3 3. Indoor Data Cleansing

4 4. Indoor Movement Analysis

5 5. Appendix

1 1. Outlines

2 2. Indoor Space Models & Applications

3 3. Indoor Data Cleansing

4 4. Indoor Movement Analysis

5 5. Appendix

1 1. Outlines

2 2. Indoor Space Models & Applications

3 3. Indoor Data Cleansing

4 4. Indoor Movement Analysis

5 5. Appendix

The End. Thanks :)