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

Huan Li

Database Laboratory, Zhejiang University

lihuancs@zju.edu.cn

April 12, 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 1. Outlines

2 2. Indoor Space Models & Applications

3 3. Indoor Data Cleansing

4 4. Indoor Movement Analysis

5 5. Appendix

About This Work...

A Foundation for Efficient Indoor Distance-Aware Query Processing. [6]
H. Lu, X. Cao, and C. S. Jensen.

- Published at *ICDE' 2012*.
- First time to propose a distance-aware indoor space model that integrates indoor distance seamlessly.
- Accompanying, efficient algorithms for computing indoor distances.
- Indexing framework that accommodates indoor distances.

Motivation

- A variety of LBS services are useful in indoor space.
    - a museum guidance service in a complex exhibition
    - boarding reminder service in an airport, to remind the passengers especially those far away from their gates or departures
- Such indoor LBSs will benefit from the availability of accurate indoor distances.
    - indoor space entities enable as well as constrain indoor movement, thus makes traditional space model for Euclidean/spatial network spaces unsuitable.
    - existing indoor space models [7, 8, 9] pay little attention to indoor distances.

Indoor Topology Mapping Structures

Mapping $D2P$ maps a door d_k to one or two partition pairs [1] (v_i, v_j) such that one can move from partition [2] v_i to partition v_j through door d_k:

[1] the basic assumption that a door corresponds to two doors can be extended by converting a door to multiple doors.

[2] a partition indicates a room, a hallway or a staircase.

Indoor Topology Mapping Structures

Mapping $D2P$ maps a door d_k to one or two partition pairs [1] (v_i, v_j) such that one can move from partition [2] v_i to partition v_j through door d_k:

$$D2P : \mathcal{S}_{door} \rightarrow 2^{\mathcal{S}_{partition}} \times 2^{\mathcal{S}_{partition}} \quad (1)$$

[1] the basic assumption that a door corresponds to two doors can be extended by converting a door to multiple doors.

[2] a partition indicates a room, a hallway or a staircase.

Indoor Topology Mapping Structures

Mapping $D2P$ maps a door d_k to one or two partition pairs [1] (v_i, v_j) such that one can move from partition [2] v_i to partition v_j through door d_k:

$$D2P : \mathcal{S}_{door} \rightarrow 2^{\mathcal{S}_{partition}} \times 2^{\mathcal{S}_{partition}} \tag{1}$$

For *enterable partition* of door d_k:

[1] the basic assumption that a door corresponds to two doors can be extended by converting a door to multiple doors.

[2] a partition indicates a room, a hallway or a staircase.

Indoor Topology Mapping Structures

Mapping $D2P$ maps a door d_k to one or two partition pairs [1] (v_i, v_j) such that one can move from partition [2] v_i to partition v_j through door d_k:

$$D2P : \mathcal{S}_{door} \rightarrow 2^{\mathcal{S}_{partition}} \times 2^{\mathcal{S}_{partition}} \tag{1}$$

For *enterable partition* of door d_k:

$$D2P_{\sqsupset} : \mathcal{S}_{door} \rightarrow 2^{\mathcal{S}_{partition}} \tag{2}$$

[1] the basic assumption that a door corresponds to two doors can be extended by converting a door to multiple doors.

[2] a partition indicates a room, a hallway or a staircase.

Indoor Topology Mapping Structures

Mapping $D2P$ maps a door d_k to one or two partition pairs [1] (v_i, v_j) such that one can move from partition [2] v_i to partition v_j through door d_k:

$$D2P : \mathcal{S}_{door} \rightarrow 2^{\mathcal{S}_{partition}} \times 2^{\mathcal{S}_{partition}} \quad (1)$$

For *enterable partition* of door d_k:

$$D2P_{\sqsupset} : \mathcal{S}_{door} \rightarrow 2^{\mathcal{S}_{partition}} \quad (2)$$

For *leaveable partition* of door d_k:

[1] the basic assumption that a door corresponds to two doors can be extended by converting a door to multiple doors.

[2] a partition indicates a room, a hallway or a staircase.

Indoor Topology Mapping Structures

Mapping $D2P$ maps a door d_k to one or two partition pairs [1] (v_i, v_j) such that one can move from partition [2] v_i to partition v_j through door d_k:

$$D2P : \mathcal{S}_{door} \rightarrow 2^{\mathcal{S}_{partition}} \times 2^{\mathcal{S}_{partition}} \quad (1)$$

For *enterable partition* of door d_k:

$$D2P_{\sqsupset} : \mathcal{S}_{door} \rightarrow 2^{\mathcal{S}_{partition}} \quad (2)$$

For *leaveable partition* of door d_k:

$$D2P_{\sqsubset} : \mathcal{S}_{door} \rightarrow 2^{\mathcal{S}_{partition}} \quad (3)$$

[1] the basic assumption that a door corresponds to two doors can be extended by converting a door to multiple doors.

[2] a partition indicates a room, a hallway or a staircase.

Indoor Topology Mapping Structures

The mapping $P2D_{\sqsupset}$ maps a partition v to all the doors through which one can enter v:

Indoor Topology Mapping Structures

The mapping $P2D_{\sqsupset}$ maps a partition v to all the doors through which one can enter v:

$$P2D_{\sqsupset} : \mathcal{S}_{partition} \rightarrow 2^{\mathcal{S}_{door}} \tag{4}$$

Indoor Topology Mapping Structures

The mapping $P2D_{\sqsupset}$ maps a partition v to all the doors through which one can enter v:

$$P2D_{\sqsupset} : \mathcal{S}_{partition} \rightarrow 2^{\mathcal{S}_{door}} \tag{4}$$

The mapping $P2D_{\sqsubset}$ maps a partition v to all the doors through which one can leave v:

Indoor Topology Mapping Structures

The mapping $P2D_{\sqsupset}$ maps a partition v to all the doors through which one can enter v:

$$P2D_{\sqsupset} : \mathcal{S}_{partition} \rightarrow 2^{\mathcal{S}_{door}} \tag{4}$$

The mapping $P2D_{\sqsubset}$ maps a partition v to all the doors through which one can leave v:

$$P2D_{\sqsubset} : \mathcal{S}_{partition} \rightarrow 2^{\mathcal{S}_{door}} \tag{5}$$

Indoor Topology Mapping Structures

The mapping $P2D_{\sqsupset}$ maps a partition v to all the doors through which one can enter v:

$$P2D_{\sqsupset} : \mathcal{S}_{partition} \rightarrow 2^{\mathcal{S}_{door}} \tag{4}$$

The mapping $P2D_{\sqsubset}$ maps a partition v to all the doors through which one can leave v:

$$P2D_{\sqsubset} : \mathcal{S}_{partition} \rightarrow 2^{\mathcal{S}_{door}} \tag{5}$$

The mapping $P2D$ is used when there's no need to differentiate the directionality:

Indoor Topology Mapping Structures

The mapping $P2D_{\sqsupset}$ maps a partition v to all the doors through which one can enter v:

$$P2D_{\sqsupset} : \mathcal{S}_{partition} \rightarrow 2^{\mathcal{S}_{door}} \tag{4}$$

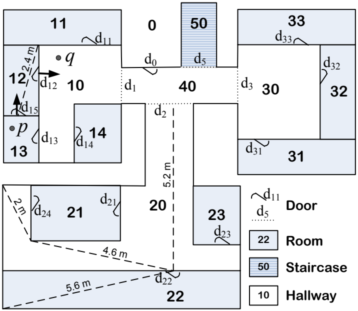
The mapping $P2D_{\sqsubset}$ maps a partition v to all the doors through which one can leave v:

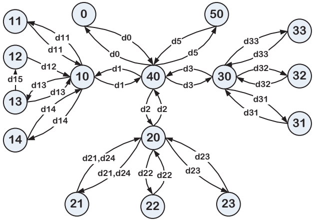
$$P2D_{\sqsubset} : \mathcal{S}_{partition} \rightarrow 2^{\mathcal{S}_{door}} \tag{5}$$

The mapping $P2D$ is used when there's no need to differentiate the directionality:

$$P2D(v_i) : P2D_{\sqsupset}(v_i) \cup P2D_{\sqsubset}(v_i) \tag{6}$$

Accessibility Base Graph





Example

$D2P_{\sqsupset}(d_{12}) = \{v_{10}\}$, $D2P_{\sqsubset}(d_{12}) = \{v_{12}\}$

$P2D_{\sqsupset}(v_{13}) = \{d_{13}\}$,

$P2D_{\sqsubset}(v_{13}) = \{d_{13}, d_{15}\}$

Accessibility Base Graph

- $G_{accs} = \{V, E_a, L\}$
- $V = \mathcal{S}_{partition}$ is the set of vertices
- $E_a = \{(v_i, v_j, d_k) | (v_i, v_j) \in D2P(d_k)\}$ is the set of labeled, directed edges
- $L = \mathcal{S}_{door}$ is the set of edge labels

Distance-Aware Model

The G_{accs} graph does not capture indoor distance information. **Extended Graph Model** is proposed to integrate indoor distances into the graph in a seamless way. *Minimum Indoor Walking Distance*(MIWD) is used.

Extended Graph Model $G_{dist} = \{V, E_a, L, f_{dv}, f_{d2d}\}$

- $V = \mathcal{S}_{partition}$ is the set of vertices
- $E_a = G_{accs}.E_a$
- $L = \mathcal{S}_{door}$ is the set of edge labels
- $f_{dv} = \mathcal{S} \times V \rightarrow \mathcal{R} \cup \{\infty\}$ maps an edge to a distance value.

$$f_{dv} = \begin{cases} \max_{p \in v_j} ||d_i, p||, & if \quad v_j \in D2P_{\sqsupset}; \\ \infty, & otherwise. \end{cases}$$

- $f_{d2d} = V \times \mathcal{S}_{door} \times \mathcal{S}_{door} \rightarrow \mathcal{R} \cup \{\infty\}$ maps a 3-tuple to a distance value.

$$f_{dv} = \begin{cases} ||d_i, d_j||_{v_k}, & if \quad d_i \in P2D_{\sqsupset}(v_k) and d_j \in P2D_{\sqsubset}(v_k); \\ \infty, & if \quad d_i = d_j and d_i, d_j \in P2D(v_k); \\ 0, & otherwise. \end{cases}$$

Computation: *door-to-door distance*

Algorithm 1 d2dDistance(Source door d_s, destination door d_t)

initialize a min-heap H
for each door $d_i \in \mathcal{S}_{door}$ **do**
    if $d_i \neq d_s$ **then**
        $dist[d_i] \leftarrow \infty$
    else
        $dist[d_i] \leftarrow 0$
    enheap$(H, \langle d_i, dist[d_i]\rangle)$
    $prev[d_i] \leftarrow$ null
while H is not empty **do**
    $\langle d_i, dist[d_i]\rangle \leftarrow$ deheap(H)
    if $d_i = d_t$ **then**
        return $dist[d_i]$
    mark door d_i as visited
    $parts \leftarrow D2P_{\sqsubset}(d_i)$
    for each partition $v \in parts$ **do**
        for each unvisited door $d_j \in P2D_{\sqsubset}(v)$ **do**
            if $dist[d_i] + G_{dist}.f_{d2d}(v, d_i, d_j) < dist[d_j]$ **then**
                $dist[d_j] \leftarrow dist[d_i] + G_{dist}.f_{d2d}(v, d_i, d_j)$
                replace d_j's element in H by $\langle d_j, dist[d_j]\rangle$
                $prev[d_j] \leftarrow (v, d_i)$

1. d_t as source door, d_s as destination door, $d2dDistance(d_t, d_s)$ finds the minimum walking distance in a $Dijkstra$ way.
2. $dist[d_j]$ stores the current shortest path distance from souce d_s to a door d_j.
3. $prev[d_j]$ stores the corresponding previous partition and door pair (v, d_i) through which the algorithm visits the current door d_j.

Computation: *point-to-point distance* (I)

Algorithm 2 pt2ptDistance(Source indoor position p_s, destination indoor position p_t)

```
v_s ← getHostPartition(p_s)
v_t ← getHostPartition(p_t)
dist ← ∞
for each door d_s ∈ P2D⊏(v_s) do
    dist_1 ← dist_V(p_s, d_s)
    for each door d_t ∈ P2D⊐(v_t) do
        dist_2 ← dist_V(p_t, d_t)
        if dist > dist_1 + d2dDistance(d_s, d_t) + dist_2 then
            dist ← dist_1 + d2dDistance(d_s, d_t) + dist_2
return dist
```

1. $getHostPartition(p)$ returns the partition that contains p.
2. $dist_V : P \times \mathcal{S}_{door} \rightarrow \mathcal{R} \cup \{\infty\}$ returns the shortest intra-partition distance between a position p and a door d, i.e., the minimum distance one must walk to get from position p to door d without leaving p's host partition.
3. minimum door-to-door distance from each door d_s in $P2D_{\sqsubset}(d_s)$ to each door $P2D_{\sqsupset}(d_t)$ is computed.
4. intra-partition distances $dist_V(p_s, d_s)$ and $dist_V(p_t, d_t)$ are added to that distance to get one possible position-to-position distance.
5. the minimum is returned as the result.

Computation: *point-to-point distance* (II)

Algorithm 3 pt2ptDistance2(Source indoor position p_s, destination indoor position p_t)

1: $v_s \leftarrow$ getHostPartition(p_s)
2: $v_t \leftarrow$ getHostPartition(p_t)
3: $doors_s \leftarrow P2D_{\sqsubset}(v_s)$
4: $doors_t \leftarrow P2D_{\sqsupset}(v_t)$
5: **for** each door $d_s \in doors_s$ **do**
6: $\quad np \leftarrow$ the partition in $D2P_{\sqsubset}(d_s) \setminus \{v_s\}$
7: $\quad$ **if** $P2D_{\sqsubset}(np) = \{d_s\}$ **and** $np \neq v_t$ **then**
8: $\quad\quad$ remove d_s from $doors_s$
9: $dist_m \leftarrow \infty$
10: **for** each door $d_s \in doors_s$ **do**
11: $\quad doors \leftarrow \emptyset$
12: $\quad$ **for** each door $d_t \in doors_t$ **do**
13: $\quad\quad$ **if** $dist_V(p_s, d_s) + dist_V(p_t, d_t) < dist_m$ **then**
14: $\quad\quad\quad$ add d_t to $doors$
15: $\quad$ initialize a min-heap H
16: $\quad$ **for** each door $d_i \in \mathcal{S}_{door}$ **do**
17: $\quad\quad$ **if** $d_i \neq d_s$ **then**
18: $\quad\quad\quad dist[d_i] \leftarrow \infty$
19: $\quad\quad$ **else**
20: $\quad\quad\quad dist[d_i] \leftarrow 0$
21: $\quad\quad$ enheap($H, \langle d_i, dist[d_i] \rangle$)
22: $\quad$ **while** H is not empty **do**
23: $\quad\quad \langle d_i, dist[d_i] \rangle \leftarrow$ deheap(H)
24: $\quad\quad$ **if** $d_i \in doors$ **then**
25: $\quad\quad\quad doors \leftarrow doors \setminus \{d_i\}$
26: $\quad\quad\quad$ **if** $dist_m > dist_V(p_s, d_s) + dist[d_i] + dist_V(p_t, d_i)$ **then**
27: $\quad\quad\quad\quad dist_m \leftarrow dist_V(p_s, d_s) + dist[d_i] + dist_V(p_t, d_i)$
28: $\quad\quad\quad$ **if** $doors = \emptyset$ **then**
29: $\quad\quad\quad\quad$ **break**
30: $\quad\quad$ mark door d_i as visited
31: $\quad\quad parts \leftarrow D2P_{\sqsubset}(d_i)$
32: $\quad\quad$ **for** each partition $v \in parts$ **do**
33: $\quad\quad\quad$ **for** each unvisited door $d_j \in P2D(v)$ **do**
34: $\quad\quad\quad\quad$ **if** $d_j \in P2D_{\sqsubset}(v)$ **then**
35: $\quad\quad\quad\quad\quad$ **if** $dist[d_i] + G_{dist}.f_{d2d}(v, d_i, d_j) < dist[d_j]$ **then**
36: $\quad\quad\quad\quad\quad\quad dist[d_j] \leftarrow dist[d_i] + G_{dist}.f_{d2d}(v, d_i, d_j)$
37: **return** $dist_m$

1. lines 1–4: $doors_s(doors_t)$ is initialized to contain all leaving(entering) doors of the source(destination) partition $v_s(v_t)$.
2. lines 5–8: a door d_s in $doors_s$ is excluded if it leads to a non-destination partition that has d_s as its sole leaving door.
3. lines 11–14: for each source door d_s, initialize a set $doors$, excluding any destination door d_t that is too far away compared to current shortest distance $dist_m$.
4. lines 15–36: follows the spirit of *Dijkstra*, only visits those doors that allow objects to move out(line 34), also updates the current shortest distance $dist_m$ when a shorter one is found.
5. the current expansion for a door d_s terminates when set $doors$ becomes empty.

Computation: *point-to-point distance* (III)

Algorithm 4 pt2ptDistance3(Source indoor position p_s, destination indoor position p_t)

1: $v_s \leftarrow$ getHostPartition(p_s)
2: $v_t \leftarrow$ getHostPartition(p_t)
3: $doors_s \leftarrow P2D_{\sqsubset}(v_s)$
4: $doors_t \leftarrow P2D_{\sqsupset}(v_t)$
5: **for** each door $d_s \in doors_s$ **do**
6: $np \leftarrow$ the partition in $D2P_{\sqsubset}(d_s) \setminus \{v_s\}$
7: **if** $P2D_{\sqsubset}(np) = \{d_s\}$ **and** $np \neq v_t$ **then**
8: remove d_s from $doors_s$
9: **for** each door $d_t \in doors_t$ **do**
10: $dists[d_s][d_t] \leftarrow \infty$
11: $dist_m \leftarrow \infty$
12: **for** each door $d_s \in doors_s$ **do**
13: $doors \leftarrow \emptyset$
14: **for** each door $d_t \in doors_t$ **do**
15: **if** $dists[d_s][d_t] = \infty$ **and** $dist_V(p_s, d_s) + dist_V(p_t, d_t) < dist_m$ **then**
16: add d_t to $doors$
17: initialize a min-heap H
18: **for** each door $d_i \in \Sigma_{door}$ **do**
19: **if** $d_i \neq d_s$ **then**
20: $dist[d_i] \leftarrow \infty$
21: **else**
22: $dist[d_i] \leftarrow 0$
23: enheap($H, \langle d_i, dist[d_i]\rangle$)
24: $prev[d_i] \leftarrow$ null

1. lines 1–8: initialize as same as the version of *pt2ptDistance2*.
2. lines 9–10: a two-dimensional array $dists[d_i][d_j]$ is employed to store the currently shortest indoor distance from source door d_i to destination door d_j.
3. line 15: the condition whether $dist[d_s][d_t]$ is infinity is check together with $dist_V(p_s, d_s) + dist_V(p_t, d_t) < dist_m$.
4. line 24: initialize $prev[d_i]$ as empty.

Computation: *point-to-point distance* (III)

25: **while** H is not empty **do**
26: $\langle d_i, dist[d_i]\rangle \leftarrow$ deheap(H)
27: **if** $d_i \in doors$ **then**
28: $doors \leftarrow doors \setminus \{d_i\}$
29: **if** $dist_m > dist_V(p_s, d_s) + dist[d_i] + dist_V(p_t, d_i)$ **then**
30: $dist_m \leftarrow dist_V(p_s, d_s) + dist[d_i] + dist_V(p_t, d_i)$
31: $(v, d_j) \leftarrow prev[d_i]$
32: **while** $d_j \neq d_s$ **do**
33: **if** $d_j \in doors_s$ and $d_j > d_s$ **then**
34: $dists[d_j][d_i] \leftarrow dist[d_i] - dist[d_j]$
35: **if** $dist_m > dist_V(p_s, d_j) + dists[d_j][d_i] + dist_V(p_t, d_i)$ **then**
36: $dist_m \leftarrow dist_V(p_s, d_j) + dists[d_j][d_i] + dist_V(p_t, d_i)$
37: $(v, d_j) \leftarrow prev[d_j]$
38: **if** $doors = \emptyset$ **then**
39: **break**
40: **else if** $d_i \in doors_s$ and $d_i < d_s$ **then**
41: **for** each door $d_j \in doors$ **do**
42: $dists[d_s][d_j] \leftarrow dist[d_i] + dists[d_i][d_j]$
43: **if** $dist_m > dist_V(p_s, d_s) + dists[d_s][d_j] + dist_V(p_t, d_j)$ **then**
44: $dist_m \leftarrow dist_V(p_s, d_s) + dists[d_s][d_j] + dist_V(p_t, d_j)$
45: **break**
46: mark door d_i as visited
47: $parts \leftarrow D2P_{\sqsubset}(d_i)$
48: **for** each partition $v \in parts$ **do**
49: **for** each unvisited door $d_j \in P2D(v)$ **do**
50: **if** $d_j \in P2D_{\sqsubset}(v)$ **then**
51: **if** $dist[d_i] + G_{dist}.f_{d2d}(v, d_i, d_j) < dist[d_j]$ **then**
52: $dist[d_j] \leftarrow dist[d_i] + G_{dist}.f_{d2d}(v, d_i, d_j)$
53: $prev[d_j] \leftarrow (v, d_i)$
54: **return** $dist_m$

1. lines 26–31: when a destination door $door_i$ is popped from priority queue H and processed, its previous door d_j on the shortest path is obtained.
2. line 32: backward optimization is continued until the current source door d_s is reached.
3. lines 33–34: if door d_j is a source door and it has not been processed by the for-loop, the shortest distance from d_j to destination door d_i is stored in $dist[d_j][d_i]$.
4. lines 35–36: shortest indoor distance from the source position to the destination is updated if necessary.
5. lines 40–45: a similar optimization also applies to the forward direction. If d_i popped from H is a source door and processed before the current for-loop, the distance can be directly used.

Indoor Distance-Aware Indexes

Definition (Door-to-Door Distance Matrix)

an N-by-N matrix, denoted as M_{d2d}, where $N = |\mathcal{S}_{doors}|$ is the total number of doors.Without loss of generality, suppose $1 \leq d_i \leq d_j \leq N$, we have:

1) $M_{d2d}[d_i, d_i] = 0$;
2) $M_{d2d}[d_i, d_j] = d2dDistance(d_i, d_j)$;
3) $M_{d2d}[d_i, d_j]$ may differ from $M_{d2d}[d_j, d_i]$ due to the directed doors.

$$
\begin{array}{cc}
 & \begin{array}{cccccc} d_1 & d_{11} & d_{12} & d_{13} & d_{14} & d_{15} \end{array} \\
\begin{array}{c} d_1 \\ d_{11} \\ d_{12} \\ d_{13} \\ d_{14} \\ d_{15} \end{array} &
\begin{pmatrix}
0 & 1.7 & 2.7 & 3.2 & 2.6 & 4.3 \\
1.7 & 0 & 1.9 & 3.4 & 3 & 4.4 \\
2.7 & 1.9 & 0 & 2 & 2.2 & 3 \\
3.2 & 3.4 & 2 & 0 & 1.2 & 1 \\
2.6 & 3 & 2.2 & 1.2 & 0 & 2.2 \\
3.2 & 3.4 & 1.5 & 3.5 & 3.7 & 0
\end{pmatrix}
\end{array}
\qquad
\begin{array}{cc}
 & \begin{array}{cccccc} \mathit{1} & \mathit{2} & \mathit{3} & \mathit{4} & \mathit{5} & \mathit{6} \end{array} \\
\begin{array}{c} d_1 \\ d_{11} \\ d_{12} \\ d_{13} \\ d_{14} \\ d_{15} \end{array} &
\begin{pmatrix}
d_1 & d_{11} & d_{14} & d_{12} & d_{13} & d_{15} \\
d_{11} & d_1 & d_{12} & d_{14} & d_{13} & d_{15} \\
d_{12} & d_{11} & d_{13} & d_{14} & d_1 & d_{15} \\
d_{13} & d_{15} & d_{14} & d_{12} & d_1 & d_{11} \\
d_{14} & d_{13} & d_{12} & d_{15} & d_1 & d_{11} \\
d_{15} & d_{12} & d_1 & d_{11} & d_{13} & d_{14}
\end{pmatrix}
\end{array}
$$

Definition (Distance Index Matrix)

an N-by-N matrix, denoted as M_{idx}. Given a door identifier d_i, two integer $1 \leq j \leq k \leq N$, $M_{d2d}[d_i, M_{idx}[d_i, j]] \leq M_{d2d}[d_i, M_{idx}[d_i, k]]$.

Indexing Indoor Moving Objects

- store objects within the same partition together in an object bucket or several chained buckets.
- a linear table **Door-to-Partition Table** (DPT) is used to store the relationship between doors and partition object bucket. Each record in DPT is a 5-tuple $(d_i, vPtr_1, dist_1, vPtr_2, dist_2)$, where d_i is a door identifier.
- if $D2P(d_i) = \{(v_j, v_k)\}$, which means that door d_i is directional from partition v_j to v_k, then field $vPtr_1$ is a null pointer while $vPtr_2$ is a pointer that points to the object bucket of partition v_k, $dist_1$ is ∞, and $dist_2$ is $G_{dist}.f_{dv}(d_i, v_k)$.

Intra-Partition Object Index and Search

- a grid index is built for spatial objects in each indoor partition.
- all spatial objects in an indoor partition v_i are organized using a corresponding bucket B_i, B_i consists of multiple sub-buckets each of which corresponds to a grid cell.
- for $rangeSearch(B_i, q, r)$, search only those grid cells that overlap the circle centered at q and with radius r, if a cell is fully inside the circle, all the objects in its sub-bucket are included directly.
- for $nnSearch(B_i, q, dist_{nn})$, search only those grid cells that overlap the circle centered at q and with radius $dist_{nn}$.

Indoor Distance-Aware Queries: *Range Query*

Algorithm 5 **range**(Position q, distance r)

1: $v \leftarrow$ getHostPartition(q)
2: $R \leftarrow$ rangeSearch(v's bucket, p, r)
3: **for** each door $d_i \in P2D_{\sqsubset}(v)$ **do**
4: $\quad r_1 \leftarrow r - dist_V(q, d_i)$
5: $\quad$ **for** j from 1 to $|\mathcal{S}_{door}|$ **do**
6: $\quad\quad d_j \leftarrow M_{idx}[d_i, j]$
7: $\quad\quad$ **if** $M_{d2d}[d_i, d_j] > r_1$ **then**
8: $\quad\quad\quad$ **break**
9: $\quad\quad$ **else**
10: $\quad\quad\quad r_2 \leftarrow r_1 - M_{d2d}[d_i, d_j]$
11: $\quad\quad\quad$ **if** DPT$[d_j].vPtr_1 \neq null$ **then**
12: $\quad\quad\quad\quad$ **if** DPT$[d_j].dist_1 \leq r_2$ **then**
13: $\quad\quad\quad\quad\quad$ add objects in DPT$[d_j].vPtr_1$'s bucket to R
14: $\quad\quad\quad\quad$ **else**
15: $\quad\quad\quad\quad\quad R \leftarrow R \cup$ rangeSearch(DPT$[d_j].vPtr_1, d_j, r_2$)
16: $\quad\quad\quad$ **if** DPT$[d_j].vPtr_2 \neq null$ **then**
17: $\quad\quad\quad\quad$ **if** DPT$[d_j].dist_2 \leq r_2$ **then**
18: $\quad\quad\quad\quad\quad$ add objects in DPT$[d_j].vPtr_2$'s bucket to R
19: $\quad\quad\quad\quad$ **else**
20: $\quad\quad\quad\quad\quad R \leftarrow R \cup$ rangeSearch(DPT$[d_j].vPtr_2, d_j, r_2$)
21: **return** R

1. a range query $Q_r(q, r)$ returns those indoor objects that are within distance r of q.
2. lines 1–2: first gets the query position q's host partition v and searches for possibly qualifying objects within v by calling $rangeSearch(v\text{'s } bucket, p, r)$.
3. lines 4–8: exploiting the index in $M_{idx}[d_i, *]$, the search is conducted in non-descending order of $M_{idx}[d_i, d_j]$, where d_j is a door covered by the distance r from position q.
4. lines 12–18: if a partition is entirely within the query range, all objects in corresponding bucket are added to the result.

Indoor Distance-Aware Queries: *Nearest Neighbor Query*

Algorithm 6 NN(Position q)

1: $nn \leftarrow$ null; $dist_{nn} \leftarrow \infty$
2: $v \leftarrow$ getHostPartition(q)
3: $(nn, dist_{nn}) \leftarrow$ nnSearch(v's bucket, $q, dist_{nn}$)
4: **for** each door $d_i \in P2D_{\sqsubset}(v)$ **do**
5:     $r_1 \leftarrow dist_V(q, d_i)$
6:     **for** j from 1 to $|\mathcal{S}_{door}|$ **do**
7:         $d_j \leftarrow M_{idx}[d_i, j]$
8:         **if** $r_1 + M_{d2d}[d_i, d_j] > dist_{nn}$ **then**
9:             **break**
10:         **else**
11:             $r_2 \leftarrow r_1 + M_{d2d}[d_i, d_j]$
12:             **if** DPT$[d_j].vPtr_1 \neq null$ **then**
13:                 $(obj, dist) \leftarrow$ nnSearch(DPT$[d_j].vPtr_1, d_j, dist_{nn} - r_2$)
14:                 **if** $dist + r_2 < dist_{nn}$ **then**
15:                     $(nn, dist_{nn}) \leftarrow (obj, dist + r_2)$
16:             **if** DPT$[d_j].vPtr_2 \neq null$ **then**
17:                 $(obj, dist) \leftarrow$ nnSearch(DPT$[d_j].vPtr_2, d_j, dist_{nn} - r_2$)
18:                 **if** $dist + r_2 < dist_{nn}$ **then**
19:                     $(nn, dist_{nn}) \leftarrow (obj, dist + r_2)$
20: **return** R

1. a nearest neighbor query $Q_{nn}(q, r)$ returns those indoor objects whose distance from q is the samllest among all objects.
2. line 2: $nnSearch(B_i, q, dist_{nn})$ searches an object bucket B_i to find the nearest neighbor from q, $dist_{nn}$ is the current shortest distance for fast prunning.
3. lines 4–19: for each door d_i through which one can leave v, search is conducted in the similar way as for range query processing.

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

References II

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

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

[7] C. Becker and F. Dürr.
On location models for ubiquitous computing.
In *Personal and Ubiquitous Computing*, pp. 20–31, 2005.

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

[9] T. Becker, C. Nagel and T. H. Kolbe
A multilayered space-event model for navigation in indoor spaces.
In *3D Geo-Information Sciences*, pp. 61–77, 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. Outlines

2. Indoor Space Models & Applications

3. Indoor Data Cleansing

4. Indoor Movement Analysis

5. Appendix

The End. Thanks :)