APPENDIX

A. ALGORITHMS

A.1 Denisty based Splitting Algorithm

Algorithm 4: DensityBasedSplitting

Input: p-sequence Θ_o, temporal distance threshold ϵ_t, spatial distance threshold ϵ_s, tolerate time span Δ_t, tolerate spatial distance Δ_s.
Output: sequence of *split* snippets $\mathcal{A}_{snpt}$.

1 time-ordered sequence $\mathcal{A}_{snpt} \longleftarrow \langle\rangle$
2 $cluster\_id \longleftarrow 0$
3 **for** each record $\theta \in \Theta_o$ **do**
4 **if** $label(\theta)$ is $null$ **then**
5 $\mathcal{N} \longleftarrow$ *RetrieveNeighbors*(θ, ϵ_t, ϵ_s)
6 $pt_m \longleftarrow$ *GetAdaptivePtm*(Θ_o, $\theta.t$, ϵ_t)
7 **if** $|\mathcal{N}| \leq pt_m$ **then**
8 $label(\theta) \longleftarrow noise$
9 **else**
10 $cluster\_id \longleftarrow cluster\_id + 1$
11 $label(\theta) \longleftarrow cluster\_id$
12 seed set $\mathcal{S} \longleftarrow \mathcal{N} \setminus \theta$
13 **for** each record $\theta' \in \mathcal{S}$ **do**
14 **if** $label(\theta')$ is $noise$ **then** $label(\theta') \longleftarrow cluster\_id$
15 **if** $label(\theta')$ is $null$ **then**
16 $label(\theta') \longleftarrow cluster\_id$
17 $\mathcal{N}' \longleftarrow$ *RetrieveNeighbors*(θ', ϵ_t, ϵ_s)
18 $pt'_m \longleftarrow$ *GetAdaptivePtm*(Θ_o, $\theta'.t$, ϵ_t)
19 **if** $|\mathcal{N}'| > pt'_m$ **then** $\mathcal{S} \longleftarrow \mathcal{S} \cup \mathcal{N}'$
20 **for** each cluster id $cluster\_id$ **do**
21 time-ordered sequence $snpt \longleftarrow \langle\rangle$
22 add all records labeled with $cluster\_id$ to $snpt$
23 mark $snpt$ as $dense$; add $snpt$ to $\mathcal{A}_{snpt}$
24 **for** any consecutive *dense* snippets $\langle snpt_i, snpt_{i+1}\rangle$ in $\mathcal{A}_{snpt}$ **do**
25 **if** *CanBeMerged*($snpt_i$, $snpt_{i+1}$, Δ_t, Δ_s) **then**
26 $\mathcal{A}_{snpt} \longleftarrow \mathcal{A}_{snpt} \setminus \langle snpt_i, snpt_{i+1}\rangle$
27 $snpt' \longleftarrow snpt_i \cup snpt_{i+1}$
28 mark $snpt'$ as $dense$; add $snpt$ to $\mathcal{A}_{snpt}$
29 **else**
30 **if** exists a record labeled with $noise$ between the snippets **then**
31 time-ordered sequence $snpt \longleftarrow \langle\rangle$
32 add all records labeled with $noise$ in-between to $snpt$
33 mark $snpt$ as non_dense; add $snpt$ to $\mathcal{A}_{snpt}$
34 **return** $\mathcal{A}_{snpt}$
35 **Function** *GetAdaptivePtm* (Θ_o, t, ϵ_t)
36 $\mathcal{N}_t \longleftarrow$ range query on Θ_o within $[t - \epsilon_t, t + \epsilon_t]$
37 $pt_m \longleftarrow \lfloor \frac{e^{|\mathcal{N}_t|-N}}{1+e^{|\mathcal{N}_t|-N}} * 2P + B \rfloor$
38 **return** pt_m

Algorithm 4 divides a cleaned p-sequence Θ_o into a sequence of data snippets. It mainly consists of a procedure of density based clustering (lines 2–19) and a procedure of p-sequence partitioning (lines 20–33).

First, a time-ordered sequence $\mathcal{A}_{snpt}$ is initialized to hold the split snippets (line 1). Subsequently, the density based clustering is conducted on the positioning records in the p-sequence Θ_o (lines 2–19). In particular, the function *RetrieveNeighbors* (lines 5 and 17) issues a range query on Θ_o to find a set $\mathcal{N}$ of neighboring records, each having its spatial distance to θ smaller than ϵ_s and temporal distance to θ smaller than ϵ_t. Besides, the function *GetAdaptivePtm* (lines 6 and 18) computes an adaptive pt_m associated with the current record.

To be specific, *GetAdaptivePtm* works as follows (lines 35–38). It first retrieves a set $\mathcal{N}_t$ of records within the time window $[t - \epsilon_t, t + \epsilon_t]$ by a temporal range query on Θ_o (line 36). Note that $|\mathcal{N}_t|$ is proportional to the local sampling rate st_{local} such that $|\mathcal{N}_t| = 2 * \epsilon_t * st_{local}$. Afterwards, a round-down sigmoid function $\lfloor \frac{e^{|\mathcal{N}_t|-N}}{1+e^{|\mathcal{N}_t|-N}} * 2P + B \rfloor$ is used to obtain the adaptive pt_m (line 37). Particularly, N, P, B are three integer parameters, and pt_m increases monotonically with $|\mathcal{N}_t|$ from B to $2P + B$ and equals to $P + B$ when $|\mathcal{N}_t| = N$.

When the clustering is done, each a cluster is converted into a dense snippet $snpt$ and added to $\mathcal{A}_{snpt}$ (lines 20–23). After that, the algorithm iterates through each consecutive dense snippets $\langle snpt_i, snpt_{i+1}\rangle$ in $\mathcal{A}_{snpt}$ (lines 24–33). It first checks if the two dense snippets $snpt_i$, $snpt_{i+1}$ can be merged in the sense of the merging conditions defined on Δ_t and Δ_s (line 25). If it is, they are removed from $\mathcal{A}_{snpt}$ (line 26), merged into one (line 27), and added back to $\mathcal{A}_{snpt}$ (line 28). Otherwise, it further checks if there exists any positioning record labeled with $noise$ between the two snippets (lines 29–30). If any of such record can be found, a non-dense snippet is formed by retrieving all such records in-between, and then added to $\mathcal{A}_{snpt}$ (lines 31–33). At the end, the time-ordered sequence $\mathcal{A}_{snpt}$ is returned (line 34).

A.2 Semantic Matching Algorithm

Algorithm 5: SemanticMatching

Input: split snippet Θ^*_o, event identification function $\mathcal{E}$, semantic region graph $G_{\mathcal{R}}$.
Output: sequence of m-semantics Λ^*_o.

1 time-ordered sequence $\Lambda^*_o \longleftarrow \langle\rangle$
2 **if** $\mathcal{E}(\Theta^*_o) =$ *stay* **then**
3 $\delta \longleftarrow$ *stay*; $\tau \longleftarrow [head(\Theta^*_o).t, tail(\Theta^*_o).t]$
4 **for** each positioning record θ_i in Θ^*_o **do**
5 $\mathcal{N}^{(i)} \longleftarrow$ find the k nearest neighboring records of θ_i
6 $conf^{(i)} \longleftarrow \left(\frac{\sum_{\theta_j \in \mathcal{N}^{(i)}} dist_I(\theta_i.l, \theta_j.l)}{|\mathcal{N}^{(i)}|}\right)^{-1}$
7 $sum\_conf \longleftarrow \sum_{\theta_i \in \Theta^*_o} conf^{(i)}$;
 $\hat{l} \longleftarrow \sum_{\theta_i \in \Theta^*_o} \frac{conf^{(i)}}{sum\_conf} \cdot \theta_i.l$
8 $\pi \longleftarrow$ search $G_{\mathcal{R}}$ for a region r that contains $\hat{l}$
9 $\lambda \longleftarrow (\pi, \tau, \delta)$; add λ to Λ^*_o
10 **else**
11 $\delta \longleftarrow$ *pass-by*; $\lambda' \longleftarrow null$
12 **for** each positioning record θ_i in Θ^*_o **do**
13 $\tau \longleftarrow [\theta_i.t, \theta_i.t]$
14 $\pi \longleftarrow$ search $G_{\mathcal{R}}$ for a region r that contains $\theta_i.l$
15 **if** $\pi = \lambda'.\pi$ **then**
16 $\lambda'.\tau \longleftarrow \lambda'.\tau \cup \tau$;
17 **else**
18 **if** λ' is not $null$ **then** add λ' to Λ^*_o
19 $\lambda' \longleftarrow (\pi, \tau, \delta)$
20 **if** $\theta_i = tail(\Theta^*_o)$ **then** add λ' to Λ^*_o
21 **return** Λ^*_o

Algorithm 5 makes use of the $\mathcal{E}$-function and the semantic region graph, to translate a snippet Θ^*_o into a sequence of m-semantics Λ^*_o.

At the beginning, a time-ordered sequence Λ^*_o is initialized to hold the m-semantics to be matched (line 1). If the snippet is associated with a stay event as indicated by the function $\mathcal{E}(\Theta^*_o)$ (line 2), it is matched to a stay m-semantics (lines 2–9). In particular, the event and temporal annotations are made at first (line 3). Subsequently, the algorithm iterates through each positioning records θ_i

and computes its location estimate confidence $conf^{(i)}$ (lines 4–6). As a result, the stay position is inferred as $\hat{l}$ (line 7) and the spatial annotation π is matched by searching on $G_{\mathcal{R}}$ (line 8). Afterwards, the matched stay m-semantics λ is added to Λ_o^* (line 9).

Otherwise, if the snippet is corresponding to a pass-by event, we conduct the matching as follows (lines 10–20). First, an event annotation *pass-by* is made for each m-semantics to be matched, and a variable λ' is used to control the merge of consecutive pass-by m-semantics (line 11). The algorithm then iterates through each positioning record θ_i (lines 12–20). Specifically, the temporal annotation is generated (line 13) and the spatial annotation is determined as the region that contains $\theta_i.l$ (line 14). If the current θ_i's spatial annotation is the same as $\lambda'.\pi$, they are merged together (lines 15–16). Otherwise, λ' can no longer be merged with any subsequent m-semantics and it is added to Λ_o^*, and the current matched annotations (π, τ, δ) is assigned to λ' (lines 17–19). Note that λ' should be added to Λ_o^* in the last loop of iteration (line 20). Finally, Λ_o^* is returned as a sequence of the matched m-semantics (line 21).

A.3 Mobility Knowledge Construction Algorithm

Algorithm 6: MobilityKnowledgeConstruction

Input: semantic region graph $G_{\mathcal{R}}$, set of *original* ms-sequences $\mathcal{S}_\Lambda$.
Output: hash table $\mathcal{MK}$ to store the mobility knowledge for each pair of stay regions.

1 hash table $\mathcal{MK} : \langle \mathcal{R} \times \mathcal{R} \rangle \rightarrow \langle P, TP \rangle$
2 **for** each *directed* pair of stay regions $\langle r_s^{\shortparallel}, r_e^{\shortparallel} \rangle$ **do**
3 $\mathcal{MK}[\langle r_s^{\shortparallel}, r_e^{\shortparallel} \rangle] \longleftarrow$ *ConstructForOnePair*$(G_{\mathcal{R}}, \mathcal{S}_\Lambda, \langle r_s^{\shortparallel}, r_e^{\shortparallel} \rangle)$
4 **return** $\mathcal{MK}$
5 **Function** *ConstructForOnePair* $(G_{\mathcal{R}}, \mathcal{S}_\Lambda, \langle r_s^{\shortparallel}, r_e^{\shortparallel} \rangle)$
6 candidate path set $P \longleftarrow$ *A*-Search*$(G_{\mathcal{R}}, r_s^{\shortparallel}, r_e^{\shortparallel})$
7 transition probability $TP : (\mathcal{R} \times \mathcal{R}) \rightarrow$ probability
8 hash table $\mathcal{H}_{PT} : PT \rightarrow$ count
9 **for** each ms-sequence Λ_o in $\mathcal{S}_\Lambda$ **do**
10 **for** each matched segment $\langle \lambda_s^{\shortparallel}, \lambda_i^{\triangleright}, \ldots, \lambda_j^{\triangleright}, \lambda_e^{\shortparallel} \rangle$ in Λ_o **do**
11 $PT \longleftarrow \langle r_i^{\triangleright}, \ldots, r_j^{\triangleright} \rangle$
12 $\mathcal{H}_{PT}[PT] \longleftarrow \mathcal{H}_{PT}[PT] + 1$
13 hash table $\mathcal{H}_s : (\mathcal{R} \times \mathcal{R}) \rightarrow$ score
14 **for** each entry $\langle PT, count \rangle$ in $\mathcal{H}_{PT}$ **do**
15 $P' \longleftarrow$ find a subset of paths that hold PT
16 **for** each path $\phi \in P'$ **do**
17 $\omega_\phi \longleftarrow L(\phi)^{-1} / \sum_{P'} L(\phi)^{-1}$
18 **for** each directly connected regions $\langle r_k, r_l \rangle$ in ϕ **do**
19 $\mathcal{H}_s[\langle r_k, r_l \rangle] \longleftarrow \mathcal{H}_s[\langle r_k, r_l \rangle] + count * \omega_\phi$
20 **for** each region r_i covered by path set P **do**
21 $Out(r_i) \longleftarrow$ find the enterable regions when leaving r_i
22 **for** each region r_j in $Out(r_i)$ **do**
23 $TP[\langle r_i, r_j \rangle] \longleftarrow \frac{\mathcal{H}_s[\langle r_i, r_j \rangle]}{\sum_{r \in Out(r_i)} \mathcal{H}_s[\langle r_i, r \rangle]}$
24 **return** $\langle P, TP \rangle$

Algorithm 6 takes $G_{\mathcal{R}}$ and a full set of ms-sequences as input, and returns the mobility knowledge in a hash table $\mathcal{MK}$.

At the beginning, the hash table $\mathcal{MK}$ is initialized to store the path set P as well as the corresponding transition probabilities TP for each *directed* pair of stay regions (line 1). For each such pair $\langle r_s^{\shortparallel}, r_e^{\shortparallel} \rangle$, a function *ConstructForOnePair* is called to obtain the corresponding P and TP (lines 2–3).

Function *ConstructForOnePair* works as follows (lines 5–24). First, it performs a *A*-Search* to find the set P of candidate paths (line 6) and constructs the transition probabilities TP for each two directly connected regions in the candidate paths (lines 7–23). In particular, a hash table $\mathcal{H}_{PT}$ that records the region patterns is constructed by processing on each ms-sequence $\Lambda_o \in \mathcal{S}_\Lambda$ (lines 8–12). Afterwards, to record the score for each pair of directly connected regions in a candidate path, a hash table $\mathcal{H}_s$ is constructed upon the candidate path set P and hash table $\mathcal{H}_{PT}$ (lines 13–19). Consequently, the transition probabilities TP is computed with the scores recorded in $\mathcal{H}_s$ (lines 20–23). At the end, the candidate path set P is returned along with TP (line 24).

A.4 M-semantics Inference Algorithm

Algorithm 7: MSemanticsInference

Input: an *observed* ms-sequence Λ_o, hash table $\mathcal{MK}$, semantic region graph $G_{\mathcal{R}}$.
Output: a *complemented* ms-sequence Λ_o.

1 **for** each observation $\langle \lambda_s^{\shortparallel}, \lambda_q^{\triangleright}, \lambda_e^{\shortparallel} \rangle \subseteq \Lambda_o$ **do**
2 $\langle P, TP \rangle \longleftarrow \mathcal{MK}[\langle r_s^{\shortparallel}, r_e^{\shortparallel} \rangle]$
3 $\hat{\phi} \longleftarrow$ *InferMostLikelyPath*$(P, TP, \langle \lambda_s^{\shortparallel}, \lambda_q^{\triangleright}, \lambda_e^{\shortparallel} \rangle)$
4 **for** each region r_x in $\hat{\phi}$ **do**
5 $\tau_x \longleftarrow$ *InferDurationTime*$(r_x, \hat{\phi}, G_{\mathcal{R}})$
6 **if** r_x has been observed in an m-semantics λ_x **then**
7 $\Lambda_o \longleftarrow \Lambda_o \setminus \lambda_x$; $\tau_x \longleftarrow \tau_x \cup \lambda_x.\tau$
8 $\Lambda_o \longleftarrow \Lambda_o \cup (r_x, \tau_x, \lambda_x.\delta)$
9 **else**
10 $\Lambda_o \longleftarrow \Lambda_o \cup (r_x, \tau_x, \textit{pass-by})$
11 **return** Λ_o

Algorithm 7 gives the procedure of inferring the missing m-semantics for an original ms-sequence, by utilizing the mobility knowledge $\mathcal{MK}$ and the indoor mobility constraints captured in $G_{\mathcal{R}}$. Given an observation $\langle \lambda_s^{\shortparallel}, \lambda_q^{\triangleright}, \lambda_e^{\shortparallel} \rangle$ between stay m-semantics $\lambda_s^{\shortparallel}$ and $\lambda_e^{\shortparallel}$ (line 1), the corresponding candidate path set P is loaded from $\mathcal{MK}[\langle r_s^{\shortparallel}, r_e^{\shortparallel} \rangle]$ (line 2), and a function *InferMostLikelyPath* is called to infer a most-likely path $\hat{\phi} \in P$ for the observation between $\lambda_s^{\shortparallel}$ and $\lambda_e^{\shortparallel}$ (line 3). Furthermore, the algorithm makes use of the GRDs between directly connected regions to infer the time period for each region r_x contained in $\hat{\phi}$ (lines 4–10). In particular, a function *InferDurationTime* is called to infer the time period τ_x for each r_x (line 5). If the current r_x has been observed in an m-semantics λ_x in Λ_o (line 6), its inferred time period is merged with λ_x to form a new m-semantics, the new m-semantics is added back to Λ_o (lines 7–8); otherwise, the algorithm maps r_x to a pass-by m-semantics and adds it to Λ_o (line 10). At the end, the complemented ms-sequence Λ_o is returned (line 11). The technical details of function *InferMostLikelyPath* and function *InferDurationTime* have been given in Section 5.2.1 and Section 5.2.2 of the paper, respectively.

B. FORMALIZATION DETAILS OF M-SEMANTICS INFERENCE

B.1 The Length of an Indoor Candidate Path

LEMMA 1. *Given an indoor path $\phi = r_i \rightarrow \ldots, \rightarrow r_j$, its path length $L(\phi) = \sum_{k=i}^{j} dist_{gr}(r_k, r_{k+1})$ is an* upper bound *of the minimum distance required to ensure any object can reach r_j from r_i through the path ϕ.*

Parameters	Settings
Query Type	TkPRQ, TkFRPQ
$\mathcal{T}$ (minutes)	60, **120**, 180, 240
k	20, 40, **60**, 80
$\|Q\|$ (% of semantic regions)	30%, **50%**, 70%

Table 5: Parameter Settings on Real Data

PROOF. Suppose that a region r_1 connects with region r_2 and the path corresponding to the GRD from r_1 to r_2 begins at a position $l_1^{(s)} \in r_1$ and ends at a position $l_2^{(e)} \in r_2$. Also, r_2 connects with region r_3 and the path corresponding to the GRD from r_2 to r_4 begins at a position $l_2^{(s)} \in r_2$ and ends at a position $l_3^{(e)} \in r_3$. Provided that an object o at the farthest position $l_1' \in r_1$ from region r_3 can go through r_2 to reach r_3 at a position $l_3' \in r_3$, and the corresponding traveling distance is $dist_I(l_1', l_2') + dist_I(l_2', l_3')$, where $l_2' \in r_2$ is a position where o went out of r_1 and entered into r_2. According to the definition of GRD, we have $\forall l_1 \in r_1$, $dist_I(l_1, l_2^{(e)}) \leq dist_I(l_1^{(s)}, l_2^{(e)})$ and $\forall l_2 \in r_2$, $dist_I(l_2, l_3^{(e)}) \leq dist_I(l_2^{(s)}, l_3^{(e)})$. Consequently, we have $dist_I(l_1', l_2') + dist_I(l_2', l_3')$
$\leq dist_I(l_1', l_2^{(e)}) + dist_I(l_2', l_3^{(e)})$
$\leq dist_I(l_1^{(s)}, l_2^{(e)}) + dist_I(l_2^{(s)}, l_3^{(e)})$
$= dist_{gr}(r_1, r_2) + dist_{gr}(r_2, r_3)$.

In a more general case, given an indoor path $r_i \rightarrow \ldots, \rightarrow r_j$, we have the minimum required distance that ensures an object can reach r_j from r_i through the path is no greater than $\sum_{k=i}^{j} dist_{gr}(r_k, r_{k+1})$. So the lemma is proved. □

B.2 Posterior Probability of an Indoor Candidate Path

Given an observed region pattern $PT^{(o)}$ and a candidate path $\phi = r_s'' \rightarrow r_a^{\triangleright} \rightarrow \ldots \rightarrow r_b^{\triangleright} \rightarrow r_q^{\triangleright} \rightarrow r_c^{\triangleright} \rightarrow \ldots \rightarrow r_d^{\triangleright} \rightarrow r_e''$, we have the posterior probability $P(\phi|PT^{(o)})$ in the context of a first-order Markov stochastic process [32] as

$$
\begin{aligned}
P(\phi|PT^{(o)}) &= P(r_s'', r_a^{\triangleright}, \ldots, r_b^{\triangleright}, r_q^{\triangleright}, r_c^{\triangleright}, \ldots, r_d^{\triangleright}, r_e''|r_s'', r_q^{\triangleright}, r_e'') \\
&= P(r_s'', r_a^{\triangleright}, \ldots, r_b^{\triangleright}, r_q^{\triangleright}|r_s'', r_q^{\triangleright}) \cdot \\
&\quad P(r_q^{\triangleright}, r_c^{\triangleright}, \ldots, r_d^{\triangleright}, r_e''|r_q^{\triangleright}, r_e'') \\
&= \frac{P(r_q^{\triangleright}|r_b^{\triangleright}) \prod_{x=a}^{b-1} P(r_{x+1}^{\triangleright}|r_x^{\triangleright}) P(r_a^{\triangleright}|r_s'') P(r_s'')}{P(r_s'') P(r_q^{\triangleright})} \cdot \\
&\quad \frac{P(r_e''|r_d^{\triangleright}) \prod_{y=c}^{d-1} P(r_{y+1}^{\triangleright}|r_y^{\triangleright}) P(r_c^{\triangleright}|r_q^{\triangleright}) P(r_q^{\triangleright})}{P(r_q^{\triangleright}) P(r_e'')} \\
&= \frac{P(r_q^{\triangleright}|r_b^{\triangleright}) \prod_{x=a}^{b-1} P(r_{x+1}^{\triangleright}|r_x^{\triangleright}) P(r_a^{\triangleright}|r_s'')}{P(r_q^{\triangleright})} \cdot \\
&\quad \frac{P(r_e''|r_d^{\triangleright}) \prod_{y=c}^{d-1} P(r_{y+1}^{\triangleright}|r_y^{\triangleright}) P(r_c^{\triangleright}|r_q^{\triangleright})}{P(r_e'')} \\
&\propto P(r_q^{\triangleright}|r_b^{\triangleright}) \prod_{x=a}^{b-1} P(r_{x+1}^{\triangleright}|r_x^{\triangleright}) P(r_a^{\triangleright}|r_s'') \cdot \\
&\quad P(r_e''|r_d^{\triangleright}) \prod_{y=c}^{d-1} P(r_{y+1}^{\triangleright}|r_y^{\triangleright}) P(r_c^{\triangleright}|r_q^{\triangleright})
\end{aligned}
$$

C. ADDITIONAL EXPERIMENTAL RESULTS

C.1 Answering Queries using M-Semantics on Real Data

In addition to the query time interval $\mathcal{T}$ we evaluated in Section 6.1.4, we also investigate the effects of other parameters related to the two top-k queries, namely k and the query set size $|Q|$.

The parameter settings are shown in Table 5, where default values are in bold. We test each parameter with others fixed to defaults. The experimental evaluations focus on the search effectiveness in terms of the metric precision (see Section 6.1.4).

Effect of k. We fix $|Q| = 202 \times 50\% = 101$ and $\mathcal{T} = 120$ miniutes, and vary k from 20 to 80. The results for TkPRQ and TkFRPQ are reported in Figure 15(a) and (b), respectively. As shown in Figure 15(a), the precisions of all search methods increase moderately with an increasing k. Since $|Q|$ is fixed in our test, a larger k tends to include more query regions in the top-k search results and therefore the precision improves in all the methods. Clearly, IMS-CAC performs the best in each k values; its precision stays higher than 0.84 with k up to 60. The results also verify the effectiveness of our cleaning method as the search methods with the cleaning (i.e., RAW-C, IMS-CA and IMS-CAC) clearly outperform other alternatives.

On the other hand, when we increase k, the precisions of all search methods stay very stable in processing the TkFRPQ query. Different from the TkPRQ, TkFRPQ needs to find k frequent region pairs from $|2^Q|$ candidate region pairs, whose number is significantly larger than that of the candidate regions in TkPRQ. In such a case, increasing k from 20 to 60 does not affect the precisions in all methods as k is relatively small compared to $|2^Q|$. Nevertheless, the m-semantics constructed by IMS-CAC are still the best in answering the TkFRPQ in different Q settings.

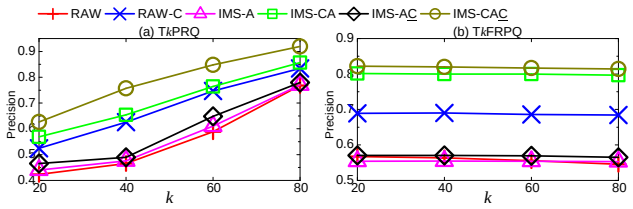

Figure 15: Query Answering Effectiveness vs. k on Real Data

Effect of $|Q|$. We also very $|Q|$ from 30% to 70% with other parameters fixed by default. The results are reported in Figure 16. Referring to Figure 16(a), increasing $|Q|$ deteriorates the TkPRQ's precision in all the methods as more query regions need to be computed and ranked. However, the precision of our overall framework IMS-CAC decreases very slightly compared to other alternatives. This shows that its constructed m-semantics are very effective to answer the top-k frequent region query. When increasing $|Q|$, the precisions of those methods without data cleaning decreases more rapidly than the alternatives that process on the cleaned data.

On the other hand, TkFRPQ's precision in each method is also insensitive to an increasing $|Q|$. In the shopping mall where our data was collected, the most frequent region pairs are always among the most popular stores visited by the shoppers. Hence, involving more semantic regions will not affect the returned results when k is fixed in the TkFRPQ query.

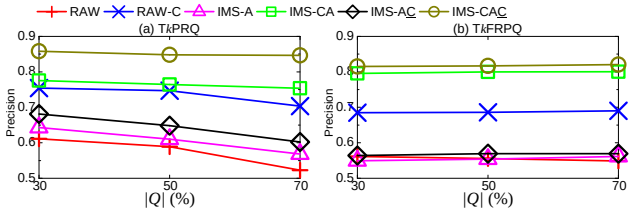

Figure 16: Query Answering Effectiveness vs. $|Q|$ on Real Data

In general, the precision results on varying k and $|Q|$ demonstrate that the m-semantics constructed by our framework IMS-CAC are very effective in answering the two indoor top-k queries.

C.2 Effectiveness of Raw Data Cleaning on Synthetic Data

According to the ground truth trajectory recorded for each object, here we study the effectiveness of our proposed raw data cleaning method on the synthetic data.

Alternative Methods. To verify the effectiveness of using the indoor mobility constraints, we consider two alternatives, namely the *ED* method that uses the Euclidean distance to represent the speed constraint as well as interpolate the new location estimates, and the *MIWD* method that uses the MIWD instead. We apply different speed threshold v_m to ED and MIWD, and compare them with the original raw p-sequence without the cleaning (denoted as *Original*).

Performance Metrics. We measure the effectiveness of ED and MIWD in two aspects. On the one hand, given a p-sequence Θ and its ground truth Θ_g, we define Θ's *floor value accuracy* (floor accuracy for short) as the fraction of its records having a correct floor value with respect to the ground truth. On the other hand, excluding those records that have false floor values, we define Θ's *average location error* (ALE) as

$$ALE = \sum_{\theta \in \Theta, \theta_g \in \Theta_g, \theta.t = \theta_g.t, \theta.l.f = \theta_g.l.f} \frac{dist_I(\theta.l, \theta_g.l)}{|\Theta|}$$

For each synthetic IPT instance (see Table 4 in the paper), we measure the average value of floor accuracies and ALEs of all p-sequences. We vary and test different settings of the maximum positioning period T and the positioning error factor μ.

Effect of T. First, we fix $\mu = 3m$ and vary T from 5s to 15s. Referring to Figure 17(a), in each setting of T, the average floor accuracy staying around 0.7 in the original p-sequence is improved significantly by our two indoor mobility constraint based methods. When increasing T, both methods' average floor accuracy decreases, but MIWD's decreases slower than ED's. The decline is due to that the time difference between consecutive records becomes larger and the speed checking tends to be less reliable. Nevertheless, when $T = 15s$, MIWD still achieves an average floor accuracy around 0.9. On the other hand, MIWD beats ED in all tests when they use the same v_m. Interestingly, when we set v_m to $1.9m/s$ and vary T from 10s to 15s, both methods' floor accuracies increase, showing that a tighter speed constraint is better to capture the object movements when the data is sparser.

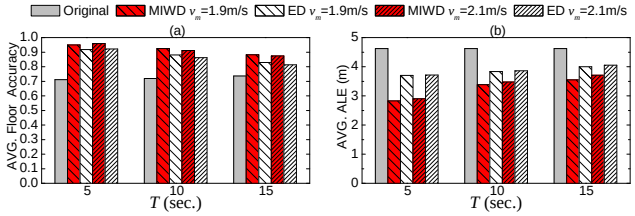

Figure 17: Cleaning Effectiveness vs. T on Synthetic Data

Referring to Figure 17(b), the average ALE of the original p-sequence is also reduced clearly by our cleaning methods. The reduction decreases but at a slow pace when a larger T is involved. Still, MIWD beats ED in all tests when we vary T.

The results reported on both measures verify that our mobility constraint based on MIWD is very effective in cleaning the raw indoor positioning data even when the data is temporally sparse.

Effect of μ. We vary and test the positioning error factor μ from 3m to 5m with T fixed to 5s. Referring to Figure 18(a), the average floor accuracy is always improved significantly in different settings of μ. Also, MIWD's improvement only decreases slightly when μ increases. As reported in Figure 18(b), the average ALE is also reduced clearly by the cleaning methods with different μs, and the reduction is more significant when μ increases. In both two measures, MIWD performs better than ED, showing it has a better capability to identify the errors and interpolate new location estimates.

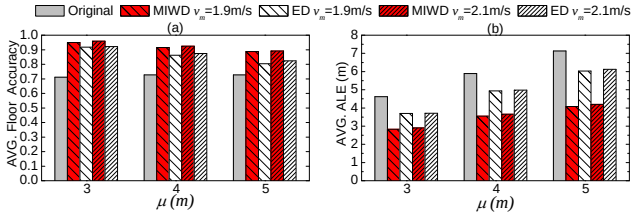

Figure 18: Cleaning Effectiveness vs. μ on Synthetic Data

To sum up, the results in different settings of T and μ verify that our indoor mobility constraint based cleaning method using MIWD is still effective when the temporal sparsity and positioning errors are involved in the raw positioning data.