Efficient Distance-Aware Query Evaluation on Indoor Moving Objects

Xie, Scott, Xike; Lu, Hua; Pedersen, Torben Bach

Published in:
Proceedings of the 29th IEEE International Conference on Data Engineering

DOI (link to publication from Publisher):
10.1109/ICDE.2013.6544845

Publication date:
2013

Document Version
Accepted author manuscript, peer reviewed version

Efficient Distance-Aware Query Evaluation on Indoor Moving Objects

Xike Xie, Hua Lu, Torben Bach Pedersen

Department of Computer Science, Aalborg University, Denmark
{*xkxie, luhua, tbp*}*@cs.aau.dk*

***Abstract*— Indoor spaces accommodate large parts of people's life. The increasing availability of indoor positioning, driven by technologies like Wi-Fi, RFID, and Bluetooth, enables a variety of indoor location-based services (LBSs). Efficient indoor distance-aware queries on indoor moving objects play an important role in supporting and boosting such LBSs. However, the distance-aware query evaluation on indoor moving objects is challenging because: (1) indoor spaces are characterized by many special entities and thus render distance calculation very complex; (2) the limitations of indoor positioning technologies create inherent uncertainties in indoor moving objects data.**

In this paper, we propose a complete set of techniques for efficient distance-aware queries on indoor moving objects. We define and categorize the indoor distances in relation to indoor uncertain objects, and derive different distance bounds that can facilitate query evaluation. Existing works often assume indoor floor plans are static, and require extensive pre-computation on indoor topologies. In contrast, we design a composite index scheme that integrates indoor geometries, indoor topologies, as well as indoor uncertain objects, and thus supports indoor distance-aware queries efficiently without time-consuming and volatile distance computation. We design algorithms for range query and k nearest neighbor query on indoor moving objects. The results of extensive experimental studies demonstrate that our proposals are efficient and scalable in evaluating distance-aware queries over indoor moving objects.

I. Introduction

Large parts of people's daily life are accommodated in various indoor spaces such as office buildings, shopping malls, conference venues, and transportation facilities, e.g., metro systems and airports. In such indoor spaces, positioning is becoming increasingly available due to different underlying technologies including Assisted GPS (A-GPS), Wi-Fi, RFID and Bluetooth. Indoor positioning provides localization for people in indoor spaces, and thus enables a variety of indoor location-based services (LBSs).

We give two examples of indoor LBSs. A cafe in a large shopping mall may send message advertisements to nearby shoppers to boost its business. A broadcasting solution for this case would be cost-inefficient and annoying to people far away. In a large airport, it is important to monitor individuals within a pre-defined range from a sensitive point, e.g., a power distribution unit. In these two examples, and in many other indoor LBS scenarios, appropriate handling of indoor distances and relevant queries is of critical importance. However, this requirement poses challenges due to several factors.

First, indoor spaces are characterized by entities such as walls, doors, rooms, etc., which render Euclidean distance and spatial network distance unsuitable [16], [24]. Such entities imply topological constraints that enable or disable movements. We show a floor plan example in Figure 1. The Euclidean distance between two points p and q does not make sense because it is blocked by a wall. To reach p from q, one has to go through doors d_{13} and d_{15} to enter room 12. One can not reach room 12 by d_{12} because the door is one-directional, as marked by the arrow. Note that one-directional doors are often seen, e.g., in security control in airports.

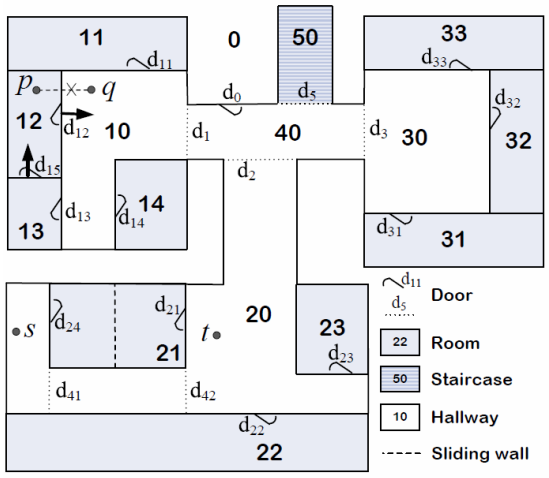


Fig. 1. Floor Plan Example

Second, indoor entities, as spatial units, can also be augmented with temporal variations. For example, a room may be temporarily available due to its opening hours, or being blocked in emergence. Also, a large room, e.g., a conference hall, may be partitioned into several smaller rooms to accommodate different events. Such reorganizations can render pre-computed indoor distances [16], [24] volatile. Refer to the example shown in Figure 1. Room 21 can be a single partition in banquet style, if the sliding wall indicated by the dashed line is dismounted. It can also be split into two partitions in meeting style if the sliding wall is mounted. Consequently, point s cannot reach t through room 21, and the distance between s and t needs recalculating by involving doors d_{41} and d_{42}.

Third, the accuracy of indoor positioning is very limited, typically varying from several to about 100 meters [1]. For example, with RFID based indoor positioning, the location of an object is reported as a region when it is in the detection

range of an RFID reader. Due to economic reasons, an indoor space is not fully covered by such readers. As a result, indoor moving objects do not get continuous location updates as their outdoor counterparts do in GPS positioning. Consequently, the location uncertainties in indoor moving objects data make it difficult to calculate object-related indoor distances.

Motivated as such, we need to support indoor distances that take into account topological constraints, temporal variations, and location uncertainties, whereas recent research [16], [24] only considers part of these important points. In this paper, we propose a complete set of techniques for efficient distance-aware queries on moving objects in realistic dynamic indoor spaces. Our proposals cover the following technical aspects that are necessary for the efficient query evaluation.

First, we define the indoor distance between a fixed point q and a moving object O, whose location is obtained through aforementioned limited indoor positioning. For the distance quantification, we choose the expected distance as it is both interpretative in entity-based queries (*range query*) and semantically comprehensive in rank-based queries (*k nearest neighbor query*) [6]. By referring to the indoor topology, we divide O's imprecise location into disjoint subregions each falling into one indoor partition (e.g., a room). Subsequently, we classify the distances between q and the subregions based on the topological properties, and derive various distance bounds that can disqualify objects in query evaluation without calculating detailed expected indoor distances.

Second, we design a composite index for indoor spaces as well as indoor moving objects, as illustrated in Figure 2. The *geometric layer* consists of a tree structure that adapts the R*-tree [3] to index all indoor partitions, as well as a skeleton tier that maintains a small number of distances between staircases. In addition, the *topological layer* maintains the connectivity information between indoor partitions, and it is implicitly integrated to the tree structure through inter-partition links. The *object layer* stores all indoor moving objects and is associated with the tree through partitions at its leaf level. Integrating the distance bounds at corresponding layers, the index supports fast pruning in query evaluation.

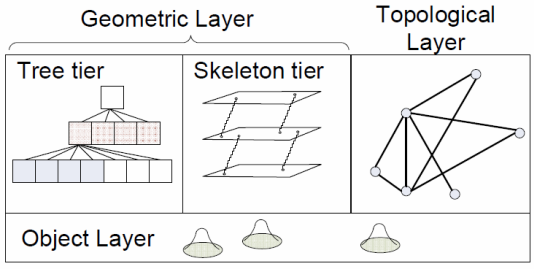


Fig. 2. Composite Index for Indoor Space

Third, we study *range query* and *k nearest neighbor query* on indoor moving objects. We define the queries by adopting the expected indoor distance, and design efficient query processing algorithms that exploit the power of the indoor distance bounds and the composite index.

We conduct extensive experiments to evaluate our proposals. The experimental results show that the indoor distance bounds and the composite index are effective, efficient and scalable for query evaluation compared to alternatives. The composite index design is also maintenance-efficient in presence of both topological updates and object updates.

Our contributions in this paper are summarized as follows.

- We study indoor distances and effective pruning bounds in relation to indoor moving objects. (Section II)
- We design a composite index for indoor spaces and moving objects. (Section III)
- We define and evaluate range queries as well as k nearest neighbor queries on indoor moving objects. (Section IV)
- We show the performance superiority of our proposals through extensive experimental studies. (Section V)

In addition, Section VI reviews the related work; Section VII concludes the paper and discusses future directions.

II. Indoor Distances for Uncertain Objects

In this section, we study indoor distances in detail. Section II-A presents preliminaries on indoor space and indoor distance. Section II-B defines the expected indoor distance for uncertain moving objects. Section II-C discusses categories of indoor distances, and Section II-D derives bounds for indoor distances. Table I lists all notations used throughout this paper.

TABLE I
Notations

Notation	Meaning
$\mathbb{O}$	a set of uncertain objects
$\mathbb{I}$, $\mathbb{E}$	Indoor space, Euclidean space
$\|p,q\|_I$	Indoor distance between p and q
$\|p,q\|_E$	Euclidean distance between p and q
$\|p,q\|_K$	Skeleton distance between p and q
$a.l$ or $a.u$	lower or upper bound of the value a
$\uparrow A$	the link/pointer to the entity A
$[R_i^-, R_i^+]$	the range for R on dimension i
$len(R_i)$	$\|R_i^+ - R_i^-\|_E$
$D(p)$	doors of partition p
$P(d)$	partitions connected to door d
$P(q)$	the partition containing point q
$P(O)$	partitions overlapping with object O
$\|O\|$	the number of instances belonging to object O
$a \overset{*d}{\rightsquigarrow} b$	a path from a to b with d as the last door
$a \overset{*}{\rightarrow} b$	the shortest path from a to b
$\odot(c,r)$	a circle centered at c with radius r

A. Preliminaries on Indoor Space and Indoor Distance

Given an indoor space, we use *partitions* to refer to rooms, staircases, or hallways. They are connected by doors or staircase entrances. For simplicity, we regard hallways and staircases as rooms. The two entrances of a staircase can be represented by doors located on the staircase's two ends. Partitions, including their associated doors, are atomic elements in indoor spaces. An indoor partition's characteristics lie in two major aspects: geometry and topology. In terms of geometry, they are 3D spatial entities in Euclidean space. Meanwhile, they are aligned to floors inside a building. For topology, partitions are separated by walls etc, and interconnected by doors or staircase entrances.

The *doors graph* [24] has been proposed to represent the connectivity of indoor partitions as well as door-to-door

distances. Formally, the doors graph is defined as a weighted graph $G_d = \langle D, E \rangle$, where:

(1) D is the set of vertices, each corresponding to a door.
(2) E is the set of edges. An edge (d_i, d_j) exists if these two doors are associated with a same partition.
(3) Each edge (d_i, d_j) has a weight that is the distance from door d_i to door d_j through their common partition.[1]

Specifically, if a door is unidirectional, i.e., allowing one-way movement only, its graph vertex's associated edges acquire directionality accordingly and are in- or out-edges. If an edge does not involve unidirectional doors, the edge is bidirectional. More details can be found in the previous work [24].

Figure 3(a) is the doors graph for the floor plan in Figure 1. One-way door d_{12}'s adjacent edges, (d_{15}, d_{12}) and (d_{12}, d_{11}), are unidirectional in the doors graph, whereas other edges that do not involve doors d_{12} or d_{15} are bidirectional, as amplified in Figure 3 (b).

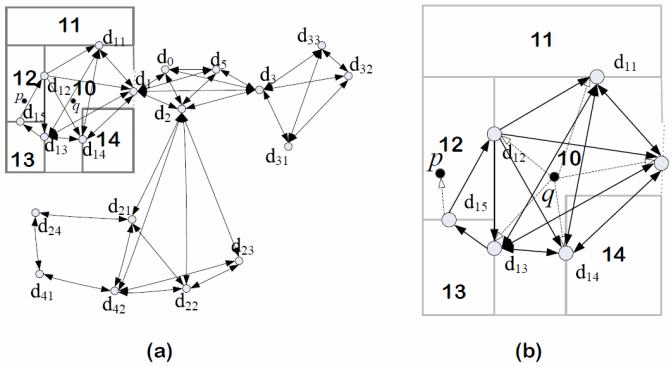


Fig. 3. Example of Doors Graph

In this paper, we do not create a separate doors graph. Instead, in our composite index for indoor space, we add extra links to the leaf-level tree nodes if their corresponding partitions are connected by a door. This design yields a de facto doors graph that is integrated in the index. More details are to be presented in Section III.

Unlike the previous works [16], [24], we do not pre-compute and store the shortest indoor distances for all door pairs before query processing. Pre-computing all such distances are expensive especially when a given indoor space has many partitions and doors. On the other hand, our decision is also justified by temporal indoor space variations we consider in this paper. As explained in Section I, partitions can be split or merged. Partitions can also be blocked in emergence or booked by sudden events, thus some doors are closed and/or temporary doors are opened accordingly. Such changes inevitably invalidate the indoor distance computing, and a considerable part of the shortest indoor distances can be affected if the temporal change happens on a pivot door or partition.

Given two indoor positions p and q, we use $q \overset{\delta}{\rightsquigarrow} p$ to denote a path from q to p where δ is the sequence of doors on that path. Referring to Figure 3(b), a path from q to p is $q \overset{d_{13},d_{15}}{\rightsquigarrow} p$ that means one can reach p from q through door d_{13} followed by d_{15}. We call the length of the shortest path as *indoor distance* from q to p, and denote it as $|q, p|_I$. Formally, $|q, p|_I = min_\delta(|q \overset{\delta}{\rightsquigarrow} p\ |)$. In the example, $q \overset{d_{13},d_{15}}{\rightsquigarrow} p$ is also the shortest path as it is the only possible path. We use $q \overset{\delta}{\rightarrow} p$ to denote the shortest path from q to p.

Indoor distance $|q, p|_I$ consists of two parts: door-door distance and intra-partition object-door distance. In Figure 3(b), door-door paths (e.g., $d_{13} \rightarrow d_{15}$) are represented by solid arrows; object-door paths (e.g., $d_{15} \rightarrow p$) are represented by dashed arrows. Let $D(p)$ be the set of doors of p's partition. In general, the indoor distance $|q, p|_I =$

$$min_{d_q \in D(q), d_p \in D(p)}(|q, d_q|_E + |d_q, d_p|_I + |d_p, p|_E) \quad (1)$$

Previous works [16], [24] assume that all possible $|d_p, d_q|_I$s are known beforehand. In this paper, we lift this assumption and investigate how to process queries without pre-computing $|d_q, d_p|_I$s. As a remark, strictly speaking, $|q, d_q|_E$ should consider the possible obstacles in q's partition. Our proposals in this paper can incorporate such obstructed distances [25] at a low level for indoor partitions. As this is not the focus of this paper, we omit the details. Reversely, the concept and the computation of obstructed distances are insufficient for modeling complex indoor topologies and distances.

B. Indoor Moving Objects and Expected Indoor Distance

Existing proposals [7], [19] model a moving object by an *uncertainty region*, where the exact location is considered as a random variable inside. The possibility of its appearance can be collected by objects' velocities [24], parameters of positioning devices [7], or analysis of historical records and thus represented by a *probability density function* (**pdf**). The pdf can be described by either a close form equation [4], [5], or a set of discrete instances [12], [15]. In this paper, we adopt the instance representation, as it is general for arbitrary distributions. Thus, an indoor moving object O is represented by a set $\{(s_i, p_i)\}$, where s_i is an instance and p_i is its *existential probability*, satisfying $\sum_{s_i \in O} p_i = 1$. Based on such probabilities, we define an expected indoor distance to measure the distance from a fixed point to an uncertain object.

Definition 1: **(Expected Indoor Distance for Uncertain Object)** Given a fixed point $q \in \mathbb{I}$ and a uncertain object O, the indoor distance from q to O is:

$$|q, O|_I = E_{s_i \in O}(|q, s_i|_I) = \sum_{s_i \in O} |q, s_i|_I \cdot p_i \quad (2)$$

In an indoor setting, an object O's uncertainty region may overlap with multiple partitions. An example is shown in Figure 6. Object O's uncertainty region overlaps with three different rooms. Accordingly, all the instances in O are divided into subsets. Generally speaking, we have $O = \cup_{1 \le j \le m} S[j]$ $(1 \le m \le |O|)$ where each $S[j]$ corresponds to a different partition and contains all those instances in that particular partition. We also call such a $S[j]$ as O's *uncertainty subregion*. We proceed to do a case study on all possible $|q, O|_I$s.

[1]The door midpoints are used for calculating door-related distances.

C. Cases of Indoor Distance $|q, O|_I$

We consider how many uncertainty subregions, i.e., $S[j]$s, object O has, and how many indoor paths exist from q to $S[j]$. Accordingly, there are three cases for $|q, O|_I$.

1) Single-Partition Single-Path Distance: In this case, O's uncertainty region falls into one single partition P. In addition, for an arbitrary $s_i \in O$, the shortest path $q \overset{*d}{\rightarrow} s_i$ shares the same door sequence ending with d through which the path enters P to reach s_i. As a result, we calculate the indoor distance as follows

$$|q, O|_I = |q, d|_I + \sum_{s_i \in O} |d, s_i|_E \cdot p_i \tag{3}$$

2) Single-Partition Multi-Path Distance: In this case, O's uncertainty region still falls into one single partition P. However, for different instances s_i and s_j, shortest paths $q \overset{*}{\rightarrow} s_i$ and $q \overset{*}{\rightarrow} s_j$ do not share the same door sequence. As a result, the indoor distance is calculated as follows

$$|q, O|_I = \sum_{s_i \in O} |q, s_i|_I \cdot p_i \tag{4}$$

An example of this case is shown in Figure 4, where O has two instances s_1 and s_2. The shortest path from q to them are: $q \overset{d_3,d_1}{\rightsquigarrow} s_1$ and $q \overset{d_2}{\rightsquigarrow} s_2$.

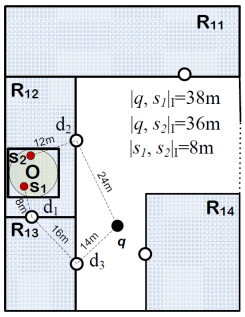


Fig. 4. $|q, O|_I$

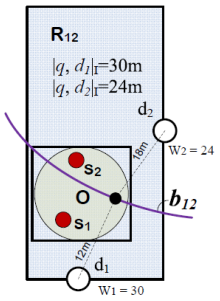


Fig. 5. Weighted Bisector b_{12}

The solution space of the single-partition multi-path distance is the *Additive Weighted Voronoi Diagram*. Suppose partition P has doors $\{d_1, \ldots, d_m\}$. For each door d_i, we assign a weight w_i, where $w_i = |q, d_i|_I$. In implementation, we can use *weighted bisectors* to represent the *Additive Weighted Voronoi Diagram*. Given two doors d_i and d_j, whose weights are w_i and w_j, respectively, the *weighted bisector* b_{ij} is a curve:

$$b_{ij} = \{p : |p, d_i|_E + w_i = |p, d_j|_E + w_j\} \tag{5}$$

There are three possible shapes for b_{ij}, as listed in Table II. Following the conditions, the bisector b_{ij} is a hyperbola, as shown in Figure 5. It splits P into two parts, such that if an instance s is on the part of d_i, the distance between q and s is calculated by path $q \overset{*d_i}{\rightarrow} s$.

In the implementation, we first check whether the bisector is null. If it exists, we further check whether an object is on a single side of the bisector. If the object intersects with the bisector, we check all its instances.

TABLE II
THE SHAPE OF b_{ij}

Shape of b_{ij}	Condition
straight line	$w_i = w_j$
hyperbola	$w_i \neq w_j$ *and* $w_i < \|d_j, P\|_{maxE}$ *and* $w_j < \|d_i, P\|_{maxE}$
null	$w_i > \|d_j, P\|_{maxE}$ *or* $w_j < \|d_i, P\|_{maxE}$

3) Multi-partition Path Distances: In this case, object O's uncertainty region overlaps with more than one partition, and thus $O = \cup_{1 \le j \le m} S[j]$ $(1 < m \le |O|)$. We calculate the indoor distance as follows

$$|q, O|_I = \sum_{1 \le j \le m} (|q, S[j]|_I \cdot \sum_{s_i \in S[j]} p_i) \tag{6}$$

In the above equation, $|q, S[j]|_I$ is calculated according to either Equation 3 or 4, by substituting $S[j]$ for O.

An example of this case is shown in Figure 6, where object O has three uncertainty subregions S_1, S_2 and S_3. Accordingly, we have $|q, O|_I = E(\sum_{1 \le j \le 3} (|q, S[j]|_I))$.

In summary, to calculate the indoor distance $|q, O|_I$, we need to find shortest paths from q to every instance $s_i \in O$. Next, we derive effective upper and lower bounds to alleviate the extensive computation.

D. Upper and Lower Bounds for Indoor Distances

Given a fixed point q and an uncertain object O in an indoor space, we derive the upper and lower bounds (*ULBounds in short*) of $|q, O|_I$ for each of the layers mentioned in Section I (see Figure 2 also). Specifically, they are *Euclidean Lower Bounds* for the geometric layer, *Topological ULBounds* for the topological layer, and *Probabilistic ULBounds* for the object layer where object probabilities are available for use.

1) Euclidean Lower Bounds: For point q and object O in an indoor space, the (virtual) Euclidean distance between them is the lower bound of their indoor space. Therefore, we have $|q, O|_{minE} \le |q, O|_I$, where $|q, O|_{minE} = min_{s_i \in O} |q, s_i|_E$.

Note that it is impossible to derive the indoor upper bounds by using Euclidean distances only. However, indoor distances can be upper bounded by a mixture of Euclidean distances and topological constraints.

2) Indoor Topological ULBounds: For point q and object $O = \cup_{i=1}^{m} S[i]$, suppose that $P(q)$ is the partition containing q, $P(S[i])$ is the partition containing $S[i]$, and $P(O)$ are the partitions overlapping with O.

Lemma 1: **(Topological LBound)** Let $t_{min}(S[i])$ be:

$$\min_{d_q \in D(P(q)), d_s \in D(P(S[i]))} |q, d_q|_E + |d_q \overset{*}{\rightarrow} d_s| + |d_s, S[i]|_{minE}$$

Then, $|q, O|_I \ge min\{t_{min}(S[i])\}$.

Lemma 2: **(Topological UBound)** Let $t_{max}(S[i])$ be:

$$\min_{d_q \in D(P(q)), d_s \in D(P(S[i]))} |q, d_q|_E + |d_q \overset{*}{\rightarrow} d_s| + |d_s, S[i]|_{maxE}$$

Then, $|q, O|_I \le max\{t_{max}(S[i])\}$.

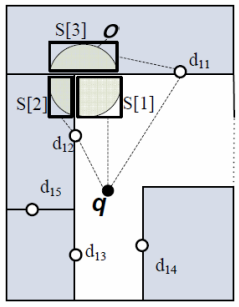


Fig. 6. Multi-Partition Path

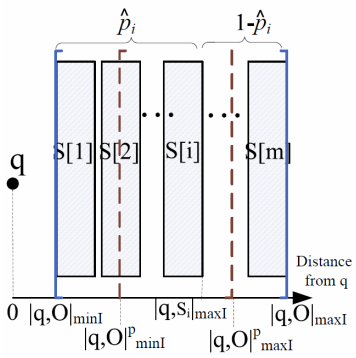


Fig. 7. Upper / Lower Bound

If object O's uncertainty region only overlaps with one partition, the above two lemmas can be simplified into:

$$\min_{d_q \in D(P(q)), d_s \in D(P(O))} |q, d_q|_E + |d_q \overset{*}{\rightarrow} d_s| + |d_s, O|_{minE} \leq |q, O|_I \leq \min_{d_q \in D(P(q)), d_s \in D(P(O))} |q, d_q|_E + |d_q \overset{*}{\rightarrow} d_s| + |d_s, O|_{maxE} \quad (7)$$

From Lemma 1 and 2, to derive the *ULBounds* requires computing shortest paths (e.g. $|d_q \overset{*}{\rightarrow} d_s|$) on the doors graph. Then, we design a looser topological upper bound. It is not as tight as *Topological UBound*, but it is more economic to be derived. Instead of getting the shortest paths, it only requires knowing some paths connecting point q and subregion $S[i]$. We call it *Topological Looser UBound*.

Lemma 3: **(Topological Looser UBound, TLU)** Let $t_{max}(S[i])$ be:

$$\min_{d_q \in D(P(q)), d_s \in D(P(S[i]))} |q, d_q|_E + |d_q \overset{*}{\rightsquigarrow} d_s| + |d_s, S[i]|_{maxE}$$

Then, $|q, O|_I \leq max\{t_{max}(S[i])\}$.

As to be detailed in Section IV, we use the looser bounds to prune doors and partitions in query processing. Afterwards, the shortest paths are only evaluated on the remaining doors and partitions for the topological ULBounds.

3) Indoor Probabilistic ULBounds: If object O overlaps with multiple partitions, the topological ULBounds may be very loose. Refer to the example shown in Figure 6, where object $O = \cup_{i=1}^{3} S[i]$. The distance from q to $S[1]$ is short, while the distance to $S[3]$ is long. If the gap between topological upper and lower bound is large, the expected distance is only constrained by a loose range but not well approximated. To tackle this problem, we design *Probabilistic Upper/Lower Bounds* by using probability information associated with objects.

Suppose object O overlaps with m partitions ($O = \cup_{i=1}^{m} S[i]$), and $S[i]$s are sorted according to the minimum distance to a given point q, as shown in Figure 7. We use $\hat{p_i}$ to denote $\sum_{j=1}^{i} p_i$. As $S[i]$ and $S[j]$ do not overlap, by using Markov Inequality, we have:

Lemma 4: **(Markov Lower Bound)**

$$E(|q, O|_I) \geq |q, S[i]|_{maxI} \cdot (1 - \hat{p_i})$$

It is possible to use *Markov Inequality* to derive an upper bound as well. However, Lemma 4 is not tight enough. Thus, we derive tighter upper/lower bounds.

Lemma 5: **(Probabilistic ULBounds)**

$$|q, S[i]|_{maxI} \cdot (1 - \hat{p_i}) + |q, O|_{minI} \cdot \hat{p_i} \leq E(|q, O|_I) \leq |q, O|_{maxI} \cdot (1 - \hat{p_i}) + |q, S[i]|_{maxI} \cdot \hat{p_i} \quad (8)$$

Proof: $E(|q, O|_I) =$

$$E(|q, \cup_{j \leq i} S[j]|_I) \cdot \hat{p_i} + E(|q, \cup_{k>i} S[k]|_I) \cdot (1 - \hat{p_i}) \quad (9)$$

Since $|q, S[i]|_{maxI} \geq E(|q, \cup_{j \leq i} S[j]|_I) \geq |q, O|_{minI}$ and $|q, O|_{maxI} \geq E(|q, \cup_{k>i} S[k]|_I) \geq |q, S[i]|_{maxI}$, we substitute them into Equation 9, and the lemma is proved. ■

An example of *Probabilistic ULBounds* is shown in Figure 7. If there are many such $S[i]$s, we prefer to choose bigger "i" to derive the lower bound and smaller "i" to derive the upper bound [2]. Although subregions are disjoint, their distance ranges may overlap. In case that no such $S[i]$ (when all $S[i]$s overlap) is found, we do not have to apply Lemma 5. If their distance ranges are very close to each other, the topological bounds are very tight. In this case, Lemma 5 still holds, but degenerates to Topological ULBounds.

E. Summary

To summarize, we use topological ULBounds for the case that an object overlaps with a single partition; and use probabilistic ULBounds for the case that an object overlaps with multiple partitions, as shown in Table III. With the ULBounds, as well as the approximate indoor distances, we avoid computing shortest paths for all existential instances of an uncertain object. However, we still need to find shortest paths for other objects and instances when using these bounds. To accelerate such shortest path computing, we design a composite index scheme to enable search space pruning.

TABLE III
INDOOR DISTANCES AND THEIR UPPER / LOWER BOUNDS

Indoor Distance	Bounds
Single-partition single-path distance Single-partition multi-path distance	Indoor Topological Upper/ Lower Bounds (Equation 7)
Multi-partition path distance	Indoor Probabilistic Upper/ Lower Bounds (Equation 8)

III. COMPOSITE INDEX FOR INDOOR SPACES

Our composite index consists of three layers, namely *Geometric layer*, *Topological layer*, and *Object layer*. The geometric layer consists of tree tier and skeleton tier. Section III-A details the composite index structure. Section III-B presents the *Geometric Lower Bound* property which is useful in the query phase. Section III-C briefly discusses index updates.

[2]Although the tightest bounds can be derived by elaborating all combinations, it costs more than calculating the exact distance itself, which contradicts our intention. For simplicity, we adopt the heuristic described above.

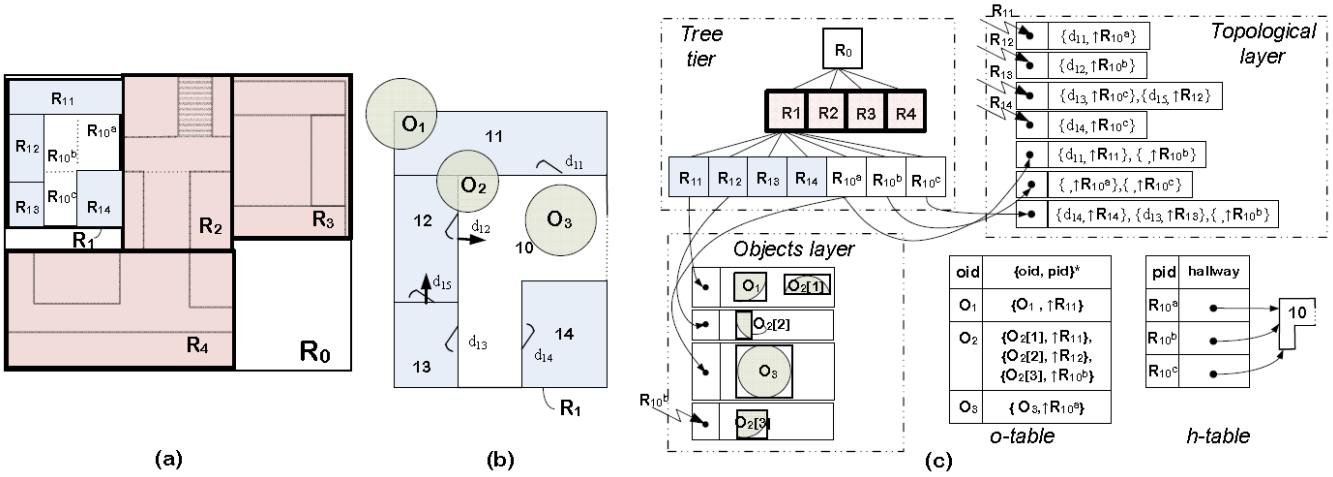


Fig. 8. An Example of the Composite Indoor Index (without the skeleton tier)

A. Composite Index Structure

1) Overview: For the floor plan shown in Figure 1, its composite index is shown in Figure 8(c). Figure 8(a) is a planar view of the index and Figure 8(b) is an amplified view of the floor plan part covered by tree node R_1.

Indoor partitions are indexed by the *Tree Tier*, called *ind*R-tree, that adapts an R-tree and treats the floor plan as an Euclidean space. Large partitions may be decomposed into small ones, each of which corresponds to a leaf node entry. Each leaf node, which represents a (sub)partition, is associated with a bucket of objects in that partition. The set of all object buckets form the *Object Layer*. This way, the object can be easily located to one or more indoor partitions given its positioning information (either a location or an uncertainty region) via the tree. Since the Euclidean distance is a lower bound of the indoor distance, the hierarchical tree structure supports indoor distance-aware queries efficiently, by pruning away disqualified candidates at higher levels.

The indoor topology information is covered by the *Topological Layer*. To support indoor distance calculation, especially for the door-to-door distance, we can traverse the topological layer in the way of traversing a graph. In addition, the *o-table* maps an object to the tree leaf nodes it overlaps with, while the *h-table* stores the mappings from a leaf node entry to an indoor partition it belongs to.

2) Tree Tier: Indoor partitions like rooms and hallways are special spatial entities. They occupy 3D regions, spanning two horizontal dimensions and one vertical dimension. Considering a building consisting of many floors, the closest facility (e.g., a restroom) might be the one upstairs. Therefore, the distance of the vertical dimension should be considered.

On the other hand, for the entities on the same floor, we care more about their planar distances. If a partition is represented by a 3D *Minimum Bounding Rectangle* (MBR in short) in *ind*R-tree, the maximum 3D distance will surely surpass its planar counterpart. This would degrade the tree's pruning performance while handling queries.

However, if the MBRs are planar rectangles, the splitting strategy for R-tree fails as the 3D volume of a tree node, expected to be minimized in R-tree construction, is always 0. To this end, when creating the tree we set the vertical length for one partition to 1 centimeter, which is very small compared to its horizontal length. Let the vertical dimension be the third dimension. We set an MBR R's vertical range to be $[R_3^-, R_3^+]$, where R_3^+ increases R_3^- by 1 centimeter. In the query phase, while calculating distances, we consider the R's vertical range to be $[R_3^-, R_3^-]$, where the vertical length is neglected. In other words, the partition is treated as a 2D rectangle distributed in the 3D space in query phase. This design gives two advantages: 1) it reduces the distance calculation workload; 2) it makes the distance reflected in the tree more accurate without the disturbance from the vertical dimension.

Some special partitions, such as a hallway, may be very imbalanced: long in one dimension but short in the other in the planar space. It may also be a non-convex region, e.g., hallway 10 in Figure 8(b). Such irregularities cause much dead space in a tree node, and thus degrade the tree's query performance. To handle them, we decompose an irregular partition into smaller but regular regions. We call such resulting regions, as well as undecomposed regular partitions, *index units*.

For an imbalanced partition, we check the ratio between the short side length and the long side length on the planar dimensions. The decomposition is done recursively on the longer dimension until that ratio of each resulting unit is no less than a given threshold T_{shape}. For a non-convex partition, we define the points at which the internal angle is greater than $180°$ as *turning points*. Then, we decompose the partition into several smaller, convex units. For a more irregularly-shaped partition, e.g., a circular shape, we use polygons to approximate it and find the turning points over the polygonized region. At each iteration of the decomposition, the turning points closer to the middle of one dimension is preferred. This would produce a more quadratic index unit, which is good for the *ind*R-tree construction and the query processing. These two criteria are implemented in Algorithm 3 in Appendix A.

For example, in the tree shown in Figure 8(a), the root node

is R_0 and the hallway 10 is decomposed into three index units: R_{10}^a, R_{10}^b, and R_{10}^c given $T_{shape} = 0.5$. The mapping between such an index unit and its original indoor partition is recorded in a hash table *h-table* when the tree is constructed. Formally,

$$h-table : \{\text{index unit}\} \rightarrow \sum\{\text{indoor partition}\}$$

In the tree tier, each leaf node represents an index unit that corresponds to either a regular, undecomposed partition or a smaller region obtained from decomposing an irregular partition. In addition to the MBRs, a leaf node also stores two types of information: 1) a linked bucket for all objects inside it; 2) links to its connected partitions. These two kinds of information belong to *Object layer* and *Topological layer*, respectively. We proceed to introduce these two layers.

3) Object Layer: Due to uncertainty, an object may overlap with multiple indoor partitions. For example, object O_2 overlaps with three partitions in Figure 8(b), namely 10, 11 and 12. In each of the three leaf-nodes' buckets, we store O_2. Meanwhile, we maintain a hash table *o-table* as follows.

$$o-table : \{O\} \rightarrow 2^{\{\text{index unit}\}}$$

Note that o-table maps an object to all the index units it overlaps, and it is tightly tied up with the tree tier. When an object update occurs, o-table needs to be updated accordingly. We discuss such updates in Section III-C.

4) Topological Layer: We maintain the connectivity between partitions in this layer. Here, to simplify the discussion, we assume each door always connects two partitions. As introduced in Section III-A.1, each leaf node stores a (sub)partition. For accessibility, we also store the doors belonging to the partition, and the links to accessible partitions through each door. Referring to the running example shown in Figure 8(c), for partition R_{12}, we store door d_{12}, together with its accessible partition's link $\uparrow R_{10}^b$.

5) Skeleton Tier: In our preliminary experiments we found that the Euclidean lower bound is too loose to be effective for indoor space queries. Although it applies to road networks [20] that are modeled as planar graphs, it falls short in indoor spaces that are more complex than planar graphs. Usually, an indoor floor's horizontal extent is much larger than its height. Consider a 20-floor building where each floor is of size 600 m $\times$ 600 m $\times$ 4 m and has four staircases each on one corner. Suppose a range query is issued for the center of the ground floor and asks for objects within 300 meters. Over 90% of the building space is covered if the Euclidean lower bound is used to constrain the search. As a matter of fact, only objects on the ground floor qualify since any path to upper floors is longer than 300 meters due to the staircase positions.

Staircases can be critical in deciding whether to expand the search to other floors or not. This motivates us to design the *Skeleton Tier* that captures all staircases in a concise way to help distance based pruning in query processing. This tier is a graph. Each staircase entrance is captured as a graph node, and an edge connects two nodes if their entrances are on the same floor *or* their entrances belong to the same staircase. The weight of an edge is the indoor distance between the two staircase entrances. For the staircase plan example in Figure 9(a), its skeleton tier is shown in Figure 9(b).

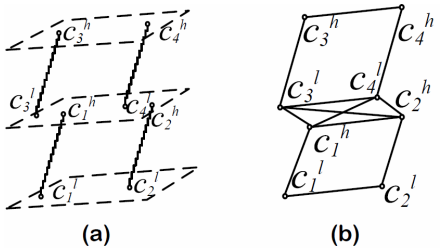


Fig. 9. Skeleton Tier Example

Let M be the total number of staircase entrances in a building, which is much smaller than that of doors in the building. We compute the indoor distance for each pair of staircase entrances and store such distances in a M by M matrix M_{s2s}. Let s_i and s_j be two staircase entrance identifiers. Matrix M_{s2s} satisfies the following properties:

(1) $M_{s2s}[s_i, s_i] = 0$;
(2) $M_{s2s}[s_i, s_j] = |s_i, s_j|_E$ if s_i and s_j are on the same floor;
(3) if s_i and s_j are of a same staircase, $M_{s2s}[s_i, s_j]$ is the shortest distance from s_i to s_j within that staircase;
(4) $M_{s2s}[s_i, s_j]$ is calculated as the shortest path distance from s_i to s_j in the skeleton layer for other cases.

B. Indoor Distance Bounds in the Geometric Layer

Within the geometric layer of the composite index, we can derive tighter indoor distance bounds than the Euclidean distance bounds. Let q be a fixed indoor point, $q.f$ the floor of q, and $S(q.f)$ all the staircases on floor $q.f$. We define the skeleton distance from two points q to p as follows.

Definition 2: **(Skeleton Distance)** Given two points p and q, their skeleton distance $|q,p|_K = |q,p|_E$ if they are on the same floor; otherwise, $|q,p|_K = \min_{s_q \in S(q.f), s_p \in S(p.f)} (|q,s_q|_E + M_{s2s}[s_q, s_p] + |s_p, p|_E)$.

If q and p are on different floors, reaching p from q has to go through one staircase entrance on q's floor and another on p's floor. Therefore, the skeleton distance sums up the Euclidean distance and the indoor distance. Hence, we define the skeleton distance as the alternative *Geometric Distance*. Now we design the *Geometric Lower Bound Property* based on that.

Lemma 6: **(Geometric Lower Bound Property)** Given two points p and q, their skeleton distance lower bounds their indoor distance, i.e., $|q,p|_K \leq |q,p|_I$.

Proof: If q and p are on the same floor, $|q,p|_K = |q,p|_E \leq |q,p|_I$. Otherwise, suppose $s_q^* \in S(q.f)$ and $s_p^* \in S(p.f)$ are on the shortest path from q to p, denoted by $q \overset{*s_q^* * s_p^* *}{\rightarrow} p$. Since $|q,p|_K = \min_{s_q \in S(q.f), s_p \in S(p.f)} (|q,s_q|_E + M_{s2s}[s_q, s_p] + |s_p, p|_E) \leq |q, s_q^*|_E + M_{s2s}[s_q^*, s_p^*] + |s_p^*, p|_E = |q,p|_I$, the lemma is proved. ■

Consider an entity e that is either an object or an *ind*R-tree node. If e spans multiple floors, we use interval $[e.lf, e.uf]$ to

represent all those floors. Note those floors must be consecutive. We define the mininum skeleton distance $|q, e|_{minK}$:

$$|q, e|_{minK} = \begin{cases} |q, e|_{minE}, \ if \ q.f \in [e.lf, e.uf]; \\ min \\ \{ \min\limits_{s_q \in S(q.f), s_e \in S(e.lf)} (|q, s_q|_E + M_{s2s}[s_q, s_e] + |s_e, e|_{minE}), \\ \min\limits_{s_q \in S(q.f), s_e \in S(e.uf)} (|q, s_q|_E + M_{s2s}[s_q, s_e] + |s_e, e|_{minE})\}, \\ otherwise. \end{cases} \quad (10)$$

With $|q, e|_{minK}$ [3], we can constrain the search via the *ind*R-tree to a much smaller range compared to if we use the Euclidean distance bounds. We design an algorithm called `RangeSearch`, as shown in Algorithm 4 in Appendix B. The algorithm takes a query point q and a distance r as input, and returns the objects and partitions within the specified range. When r=0, the query degenerates to a point-location query that returns the partition containing q.

C. Dynamic Operations on the Index

Both indoor topology and objects have dynamic natures as aforementioned. We proceed to introduce the update operations on the two layers, as well as corresponding adjustments to the tree tier.

1) Topological Layer Operations: We design update operations for partitions. As doors are associated with partitions, their operations are contained by partitions.

Insertion. When the topological change leads to a new indoor partition P, P (or its sub-partitions due to decomposition for irregularity) is inserted into the *ind*R-tree, its leaf node is connected to the adjacent partitions, and the h-table is updated if a decomposition is invovled.

Deletion. If a partition P is to be deleted, it is removed from the *ind*R-tree, the links involving P are removed from the adjacent partitions, and P's entry in the *h-table* is deleted if P is a sub-partition due to a decomposition.

If a change involves a staircase s, we need to update M_{s2s}. However, we only need to update the distances involving the staircase entrances to s and those on the same floor as s.

2) Object Layer Operations: We consider object insertion and deletion, as an object update can be implemented as a deletion followed by an insertion.

Insertion. In order to insert an object O, we search the *ind*R-tree to find the leaf nodes $\{P_i\}$ that overlap with O's uncertainty region. For each P_i, its associated buckets in the object layer is updated accordingly. Also, a new entry $\langle O, \{P_i\}\rangle$ is inserted into the o-table.

Deletion. To delete an object O, we use the o-table to find the *ind*R-tree leaf nodes $\{P_i\}$ that overlap with O's uncertainty region. For each P_i, O is removed from its associated bucket. Also, the entry for O is deleted from the o-table.

In reality, an object O comes to a partition only from its adjacent partitions. We can make use of this to speed up object updates. To simplify the presentation, we suppose O is in partition P before the update. If the location reporting in the indoor positioning/tracking is sufficiently frequent, after the update O must be stored in the bucket of P's adjacent partitions. Such partitions and their leaf nodes can be easily found by looking at the o-table, whereas search for P can be facilitated by those links in the topological layer. This way avoids searching the *ind*R-tree.

[3]Note that if e is a descendant of E ($e \subseteq E$), we have $|q, E|_{minK} \leq |q, e|_{minK}$, because one has to go through some parts of E to reach e.

IV. Efficient Query Evaluation

In this section, we study two representative queries in indoor applications: *Indoor Range Query* and *Indoor k Nearest Neighbor Query*. We define their query semantics in Section IV-A, and elaborate on query evaluation in Section IV-B.

A. Query Semantics

Definition 3: **(Indoor Range Query, iRQ)** Given a query point $q \in \mathbb{I}$ and a distance value r, the *iRQ* returns objects whose indoor distances are smaller than r. Formally, $iRQ_{q,r}(\mathbb{O}) = \{O \mid |q, O|_I \leq r, \quad O \in \mathbb{O}\}$.

Definition 4: **(Indoor k Nearest Neighbor Query, ikNNQ)** Given a query point $q \in \mathbb{I}$ and a parameter k, the *ikNNQ* returns k objects whose indoor distances to q are the smallest among all objects. Formally, $ikNN_{q,k}(\mathbb{O}) = \{O \mid O \in \mathbb{O}\}$, where $|ikNN_{q,k}(\mathbb{O})| = k$, $\forall O_i \in ikNN_{q,k}(\mathbb{O}), \forall O_j \in \mathbb{O} \setminus ikNN_{q,k}(\mathbb{O})$, $|q, O_i|_I \leq |q, O_j|_I$.

B. Efficient Query Evaluation

We make use of the indoor distances (Section II) and the index (Section III) to efficiently evaluate the iRQ and ikNNQ queries. Our query evaluation consists of 4 phases. The first phase, *filtering phase*, locates the source partition that contains the query point and retrieves candidate partitions as well as candidate objects. The second phase, *subgraph phase*, constructs a subgraph based on candidate partitions, and uses the doors of the source partition as sources to compute the shortest indoor paths that are to be used in the subsequent two phases. In the third phase, *pruning phase*, upper/lower distance bounds for objects are calculated to further reduce the number of candidate objects. In the fourth phase, *refinement phase*, the indoor distances for the remaining objects are computed and the qualifying objects are returned as the query results. We proceed to present the algorithms for iRQ and ikNNQ.

1) iRQ: The evaluation of iRQ is formalized in Algorithm 1. In the filtering step, iRQ calls `RangeSearch` (Algorithm 4 in Appendix C) to search the geometric layer. In particular, it retrieves all those objects (in R^o) and indoor partitions (in R^p) whose geometric lower bound distances (Equation 10) are no larger than the query range r. Given the *Geometric Lower Bound Property* (Lemma 6), R^o and R^P guaranteed to avoid false negatives. Specifically, any discarded entity e (object or partition) satisfies $|q, e|_I \geq |q, e|_{minK} > r$.

Set R^o is a superset of the final result. So iRQ continues to subsequent phases to verify the candidates incrementally. Specifically, the *Dijkstra* Algorithm is called to calculate single-source shortest paths starting at doors of the partition

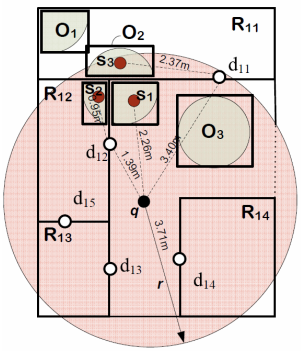


Fig. 10. iRQ

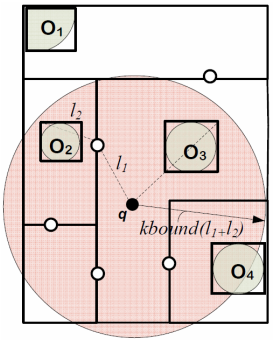


Fig. 11. ikNNQ ($k = 2$)

containing q [4] . The distance calculation only involves the partitions in R^p rather than the original topological layer that contains significantly more doors and partitions. After that, iRQ makes use of the topological upper/lower bounds to approximate indoor distances and compare them to r (`Lines 5-10`). The exact indoor distances are only computed for those objects whose bounds cover r (`Lines 11-13`).

An example of iRQ is shown in Figure 10. The circle $\odot(q, r)$ is the query region represented in the Euclidean space. Object O_1 is pruned away in filtering phase, since $|q, O_1|_{minK} > r$. After deriving the upper/lower bounds for the remaining objects in the pruning phase, O_3 is qualified. For the undetermined object O_2, the exact indoor distance is calculated and compared to r.

Algorithm 1 iRQ

```
1: function IRQ(query point q, distance r, indoor index T)
2:     result set R; candidate object set C;
3:     (R^o, R^p) ← RangeSearch(q, r, T);   // Phase1 : filtering
4:     Dijkstra(R^p);   // Phase 2: subgraph
5:     for each object O in R^o do   // Phase 3: pruning
6:         [O.l, O.u] ← [|q, O|_minI, |q, O|_maxI];   // (Table III)
7:     for each O ∈ R^o do
8:         if O.u ≤ r then R = R ∪ {O}
9:         else
10:            if O.l ≤ r then C = C ∪ {O}
11:    for each O ∈ C do   // Phase 4: refinement
12:        Calculate |q, O|_I;
13:        if |q, O|_I ≤ r then R = R ∪ {O};
14:    return R.
```

2) ikNNQ: The evaluation of ikNNQ is formalized in Algorithm 2. In the filtering step, ikNNQ first calls *kSeedsSelection* (Algorithm 5 in Appendix C) to return an object set R_1^o and a partition set R_1^p. Specifically, R_1^o contains k objects that are in query point q's partition or in the closest adjacent partitions, and R_1^p is the set of all those involved partitions. Then, ikNNQ derives *Topological Looser Upper Bounds* for the k objects and choose the longest one as *kbound* = $\max_{seed_i \in R_1^o} \{|q, seed_i|_I.TLU\}$. Next, a range search $\odot(q, kbound)$ is done on the tree tire (`Line 5`). The *Geometric Lower Bound Property* (Lemma 6) ensures zero false negatives.

[4]The weight of an edge is the Euclidean distance of two accessible doors.

An example is shown in Figure 11, where *kSeedsSelection* finds O_2 and O_3 as seeds. Because O_2's topological looser upper bound is longer, it is chosen as the kbound. Through the range search, O_1 is excluded since $|q, O_1|_K > kbound$.

The process of applying the *Dijkstra* Algorithm and deriving upper/lower bounds (`Lines 6 to 8`) are similar to iRQ. The remaining objects are sorted and O_k whose upper bound is the k-th shortest is found. Objects with $O.u$ closer than $O_k.l$ are added to the final result (`Line 11`). Objects with $O.l$ farther than $O_k.u$ have no chances as there already are k objects closer (`Line 13`). For undetermined objects, their indoor distances are calculated and the qualifying ones are picked (`Lines 14-15`). Finally, k objects having the shortest distances are returned.

Algorithm 2 ikNNQ

```
1: function IKNNQ(query point q, k, indoor index T)
2:     result set R; candidate object set C;
3:     (R_1^o, R_1^p) ← kSeedsSelection(q, k);   // Phase 1: filtering
4:     kbound ← max_{O∈R_1^o}{|q, O|_I.TLU}; //  (Lemma 3)
5:     (R_2^o, R_2^p) ← RangeSearch(q, kbound, T);
6:     Dijkstra(R_2^p);   // Phase 2: subgraph
7:     for each object O in R_2^o do   // Phase 3: pruning
8:         [O.l, O.u] ← [|q, O|_minI, |q, O|_maxI];   // (Table III)
9:     Find object O_k which has the k-th shortest O.u; set C = ∅;
10:    for each O ∈ R_2^o do
11:        if O.u < O_k.l then R = R ∪ {O}
12:        else
13:            if O.l ≤ O_k.u then C = C ∪ {O}
14:    for each O ∈ C do   // Phase 4: refinement
15:        Calculate |q, O|_I;
16:    Sort objects in C by |q, O|_I in ascending order and add top
       k − |R| objects to R;
17:    return R.
```

V. Experimental Studies

We conduct experimental studies to evaluate our proposals. Section V-A describes the experiment settings, where default parameters are bolded. Section V-B reports the results.

A. Experimental Setup

Indoor Space. We use a real floor plan of a shopping mall[5]. Each floor takes 600 m ×600 m × 4 m, with 100 rooms and 4 staircases. To test scalability, we use the plan to generate buildings with 10, **20**, and 30 floors. All of them are connected by hallways and staircases. We vary the number of floors and therefore also the number of partitions and doors.

Indoor Moving Objects. We generate a series of datasets, containing 10K, **20K**, and 30K objects randomly distributed in a given building. Objects' uncertainty regions are represented by circles, with radii 5, **10**, and 15 meters. The *pdf* is represented by a set of 100 sampling points, following Gaussian distribution. The mean is the circle center and the variance is the square of 1/6 of its diameter.

[5]http://fc06.deviantart.net/fs28/f/2008/143/4/6/Floor_Plan_for_a_Shopping_Mall_by_mjponso.png

Tree Tier. We use a packed R*-tree [17] to index all indoor partitions. The entire tree is accommodated in the main memory. We set the tree fanout to be 20, according to the results reported elsewhere [9].

Queries. Query points are randomly generated in a given building. For *i*RQ, we set the query range to 50, **100**, and 150 meters. For *i*kNNQ, we set k to 50, **100**, and 150. In all experiments, we issue 50 queries and report the average response time for each query type.

All programs were implemented in C++ and run on a Core2 Duo 3.40GHz PC enabled by MS Windows 7 Enterprise.

B. Experimental Results

Sections V-B.1 and V-B.2 report the query performances for *i*RQ and *i*kNNQ, respectively. For both query types, we test their efficiency and scalability with respect to the number of objects ($|\mathbb{O}|$), the size of uncertainty regions, and the number of partitions. Section V-B.3 investigate the effectiveness of our indoor distance bounds in filtering and pruning phases. Section V-B.4 evaluates the composite indoor index.

1) Performance of iRQ: The results of *i*RQ execution time are reported in Figure 12. Referring to Figure 12(a), the query time increases stably with $|\mathbb{O}|$ and the size of query ranges.

We show the query time break-down for default settings in Figure 12(b). The filtering and subgraph phases depend on the topologies, and thus they do not change as $|\mathbb{O}|$ increases. On the other hand, larger $|\mathbb{O}|$s make the refinement phase handle more objects that pass the filtering and pruning phases, and thus increase the query time.

As objects' uncertainty regions become larger, more objects are involved in the *i*RQ execution, and therefore the query time also increases, as shown in Figure 12(c).

We also fix the number of objects and vary the number of partitions to see the effect on query time. The results are shown in Figure 12(d). Since the average number of objects in one partition (i.e., object density in each partition) decreases, we see query time decreases accordingly.

2) Performance of ikNNQ: The results of *i*kNNQ execution time are reported in Figure 13. Referring to Figure 13(a), the query time increases stably as the number of objects and k increase. The query time break-down for default settings is shown in Figure 13(b). Compared to *i*RQ, *i*kNNQ need to retrieve more indoor partitions to find sufficient $k = 100$ candidates in the filtering phase. Consequently, the subsequent phases get higher workloads to process.

The results on the effect of object uncertainty region size are shown in Figure 13(c). Larger uncertainty sizes render more objects and partitions to be retrieved in the range search step, and thus increase the query execution time.

The results on the effect of the number of partitions are shown in Figure 13(d). Again, query time decreases as the object density in each partition decreases.

3) Effectiveness of Indoor Distance Bounds: Our indoor distance bounds contribute to the efficiency of query execution through filtering and pruning phases, as indicated by the results shown in Figure 14. We define the term *pruning ratio* as the ratio of objects disqualified over $|\mathbb{O}|$.

Referring to Figure 14(a), over 97.3% objects are filtered out by the skeleton distance bound in the filtering phase of *i*RQ in all tested settings. The results show that the skeleton layer and the skeleton distance bound are very effective in filtering indoor partitions (less than 2.7% partitions are retrieved) and objects at a high level without the search going down to the object layer. Without the filtering phase, all indoor partitions would be involved in the shortest path computation, which would be too expensive for the query execution. After the pruning phase over 99.4% objects in total are pruned.

We further study the effect of the pruning phase by including and excluding it in *i*RQ execution. The results are shown in Figure 14(b). Clearly, the topological distance bounds (Table III) used in the pruning phase are very effective in speeding up the query processing.

The counterpart results for *i*kNNQ are shown in Figures 14(c) and 14(d). Again, indoor distance bounds are very effective in discarding objects. In particular, the pruning phase contributes more for *i*kNNQ than for *i*RQ. Referring to Figure 14(d), the query time would increase by at least 4 times without the pruning phase. As discussed in Section V-B.2, *i*kNNQ involves more indoor partitions after the filtering phase and thus the topological distance bounds exert more effective influence in the pruning phase.

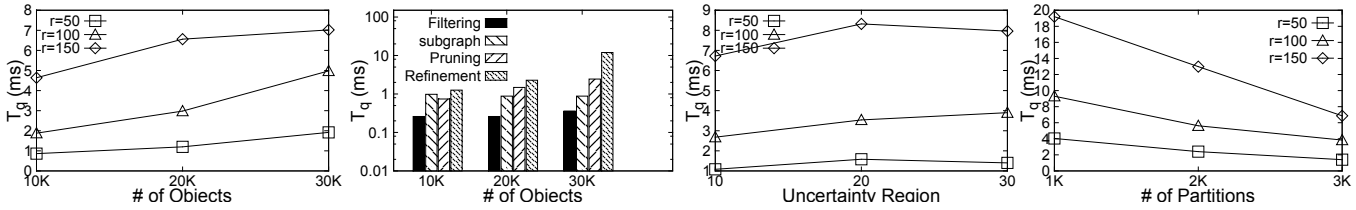
4) Composite Indoor Index: We also enable and disable the skeleton tier and compare the corresponding performances to evaluate its effectiveness. The results on the number of indoor partitions retrieved are reported in Figure 15 (a). Clearly, the skeleton tier effectively supports partitions' retrieval, e.g., almost two thirds irrelevant partitions are excluded when the query range equals to 100.

We evaluate the construction time for the composite index and report the results in Figure 15 (b). In all tested cases, the construction finishes within several seconds. Note that the skeleton tier is constructed in only one millisecond.

We also study the time costs for update operations on the composite index and report the results in Figure 15(c). With 2000 partitions in an indoor building, each object update operation costs only less than 0.01 milliseconds on average, whereas each topology update operation costs less than 1 milliseconds on average. In contrast, distance pre-computation is very expensive, as shown in Figure 15(d). With the same 2000 partitions, more than half an hour is needed to update the door-to-door distances in case of a topological change in the indoor space. This significant performance gap justifies our composite index design without door-to-door distance pre-computation. Our composite index for indoor spaces is an update-efficient design.

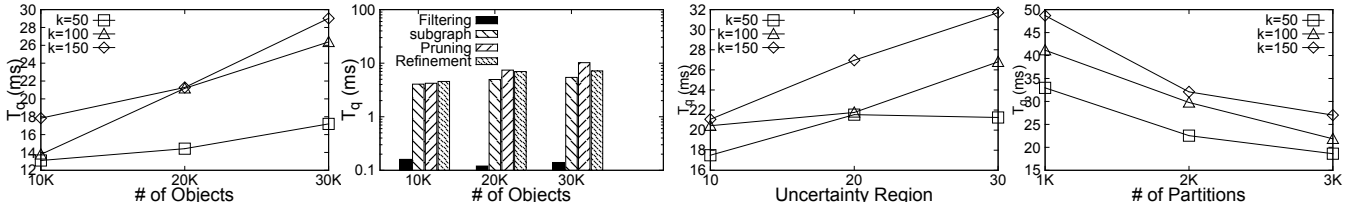
VI. Related work

Different indoor space models have been proposed. The 3D Geometric Network Model [13] treats the vertical and horizontal connectivity relationship among 3D spatial cells separately. The 3D Indoor Geo-Coding technique employs



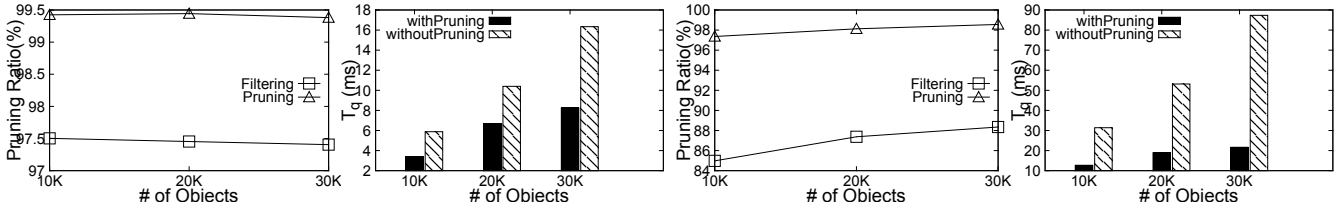
(a) T_q vs. $|\mathbb{O}|$ (b) T_q's Breakdown (c) T_q vs. Uncertainty (d) T_q vs. # of Partitions

Fig. 12. *i*RQ Query Execution Time



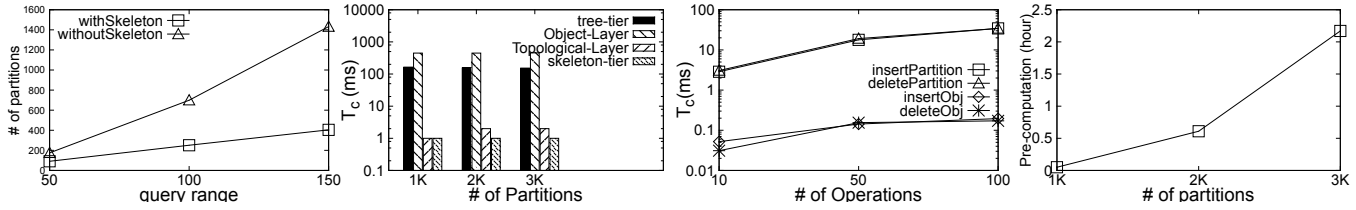
(a) T_q vs. $|\mathbb{O}|$ (b) T_q's Breakdown (c) T_q vs. Uncertainty (d) T_q vs. # of Partitions

Fig. 13. *i*kNNQ Query Execution Time



(a) Filtering & Pruning (*i*RQ) (b) Pruning Phase (*i*RQ) (c) Filtering & Pruning (*i*kNNQ) (d) Pruning Phase (*i*kNNQ)

Fig. 14. Effectiveness of Indoor Distance Bounds



(a) Skeleton Distance (b) T_c vs. Partitions (c) T_c vs. Dynamic Operations (d) Distance Pre-computation Time

Fig. 15. Results for Composite Indoor Index

the 3D *Poincaré Duality* [18] to transform 3D spatial cells from primal space to dual space. A 3D metrical-topological model [22] describes both the shapes and connectivity of spatial cells for navigation purposes. Another 3D model [2] combines space partitions with possible events in a dual space, to enable navigation in multi-layered buildings. Focusing on topological relationships, these models do not support indoor distances and relevant queries.

A lattice-based semantic location model [14] defines the "length" of an indoor path by the number of doors on the path rather than the actual indoor distance. As a result, this model falls short in many practical scenarios [16]. Different ways of transforming a floor plan into a graph also exist [8], [10], [21], but such proposals lack support for indoor distances.

Research on indoor moving objects often assumes symbolic indoor space modeling and indoor positioning [10]. R-tree based structures [11] have been used to index offline trajectories of moving objects in symbolic indoor spaces. By differentiating object states in terms of positioning detection, a hash indexing method [23], [24] has been designed to index the online positions of indoor moving objects.

Previous works [16], [23], [24] study spatial queries on online indoor moving objects. This paper differs from these works in several aspects. First, the range query definition in this paper employs indoor distance while the previous work [23] focuses on semantic units, e.g., a room or a positioning device, as query ranges and does not support indoor distances. Second, the k nearest neighbor query in this paper returns the top-k uncertain indoor moving objects with the shortest expected indoor distances to the query point. With a different semantic interest, the nearest neighbor query in previous work [24] returns all k-subsets of objects whose collective probability of being the k nearest neighbors, in terms of the minimum indoor walking distance to the query point, is higher than a pre-defined threshold. Note that this paper does not employ such a probability threshold. Third, previous works [16], [24] assume that all door-to-door distances are pre-computed and available for query processing, whereas this

paper lifts this assumption and computes indoor distances on the fly in query processing. Fourth, the previous work [16] queries on indoor static objects (points of interest, i.e., POIs) while the queries in this paper are on indoor moving objects.

VII. Conclusion and Future Work

In this paper, we study efficient evaluation of distance-aware queries on indoor moving objects. We investigate the indoor distance categories regarding object location uncertainties and indoor topologies. To speed up distance based pruning in query evaluation, we propose effective indoor distance upper/lower bounds. We also design a composite index for indoor space as well as objects, which facilitates efficient indoor distance retrieval as well as query processing. Extensive experimental results demonstrate that our proposals are effective, efficient and scalable in various query settings. In addition, our composite index incurs significantly less maintenance costs compared to a pre-computation based alternative.

Our work in this paper opens directions for future work. First, it is of interest to study other query types using the distance bounds and the composite index proposed in this paper. Second, it is useful to estimate the selectivity for indoor distance aware queries and make use of it in further optimizing queries over uncertain objects. Third, it is beneficial to reuse computational efforts on indoor distances when multiple, related queries are issued within a short period of time.

Acknowledgments

This work is partially supported by the BagTrack project funded by The Danish National Advanced Technology Foundation under Grant No. 010-2011-1.

References

[1] Accurate indoor positioning and navigation at the threshold. http://www.loctronix.com/news/insider/i1-1-a3-scpdain.html (accessed July 2012).

[2] T. Becker, C. Nagel, and T. H. Becker. A multilayered space-event model for navigation in indoor spaces. *Proc. 3rd International Workshop on 3D Geo-Info*, 2008.

[3] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In *SIGMOD*, 1990.

[4] R. Cheng, D. Kalashnikov, and S. Prabhakar. Evaluating probabilistic queries over imprecise data. In *SIGMOD*, 2003.

[5] R. Cheng, D. V. Kalashnikov, and S. Prabhakar. Querying imprecise data in moving object environments. *TKDE*, 16(9), 2004.

[6] G. Cormode, F. Li, and K. Yi. Semantics of ranking queries for probabilistic data and expected ranks. In *ICDE*, pages 305–316, 2009.

[7] D.Pfoser and C. Jensen. Capturing the uncertainty of moving-objects representations. In *SSDBM*, 1999.

[8] G. Franz, H. Mallot, J. Wiener, and K. Neurowissenschaft. Graph-based Models of Space in Architecture and Cognitive Science-a Comparative Analysis. In *IIAS InterSymp*, pages 30–38, 2005.

[9] S. Hwang, K. Kwon, S. Cha, and B. Lee. Performance evaluation of main-memory r-tree variants. In *STD*. 2003.

[10] C. Jensen, H. Lu, and B. Yang. Graph model based indoor tracking. In *MDM*, 2009.

[11] C. S. Jensen, H. Lu, and B. Yang. Indexing the trajectories of moving objects in symbolic indoor space. In *SSTD*, pages 208–227, 2009.

[12] H.-P. Kriegel, P. Kunath, and M. Renz. Probabilistic nearest-neighbor query on uncertain objects. *DASFAA*, 2007.

[13] J. Lee. A spatial access-oriented implementation of a 3-d gis topological data model for urban entities. *GeoInformatica*, 8(3):237–264, 2004.

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

[15] X. Lian and L. Chen. Monochromatic and bichromatic reverse skyline search over uncertain databases. In *SIGMOD*, 2008.

[16] H. Lu, X. Cao, and C. Jensen. A foundation for efficient indoor distance-aware query processing. In *ICDE*, 2012.

[17] M.Hadjieleftheriou. Spatial index library version 0.44.2b.

[18] J. Munkres. *Elements of algebraic topology*. Addison Wesley Publishing Company, 1993.

[19] P. Sistla et al. Querying the uncertain position of moving objects. In *Temporal Databases: Research and Practice*. 1998.

[20] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao. Query processing in spatial network databases. In *VLDB*, 2003.

[21] C. van Treeck and E. Rank. Analysis of building structure and topology based on graph theory. In *ICCCBE*, 2004.

[22] E. Whiting, J. Battat, and S. Teller. Topology of urban environments. In *CAAD Futures*, pages 115–128, 2007.

[23] B. Yang, H. Lu, and C. Jensen. Scalable continuous range monitoring of moving objects in symbolic indoor space. In *CIKM*, 2009.

[24] B. Yang, H. Lu, and C. Jensen. Probabilistic threshold k nearest neighbor queries over moving objects in symbolic indoor space. In *EDBT*, 2010.

[25] J. Zhang, D. Papadias, K. Mouratidis, and M. Zhu. Spatial queries in the presence of obstacles. In *EDBT*, pages 366–384, 2004.

Appendix

A. Decomposition (Algorithm 3)

Algorithm 3 Decompose

function DECOMPOSE(Region r, a set of turnning points P, threshold T_{shape})
  if r is concave **then**
    let $R(r)$ be the **MBR** of r;
    select a *turning point* $t \in P$ on r's boundary, such that t is closer to the middle of r;
    draw a splitting line perpendicular to the longer dimension d to divide r into two or more regions: $\{r_i\}$;
    for each r_i in $\{r_i\}$ **do**
      `Decompose(`r_i`,` $P - \{t\}$`,` T_{shape}`);`
  else
    if $\frac{len(R(r)_1)}{len(R(r)_2)} > T_{shape}$ or $\frac{len(R(r)_1)}{len(R(r)_2)} < T_{shape}$ **then**
      find the middle point m on r's longer dimension d;
      draw a splitting line perpendicular to d to divide r into two regions: r_1 and r_2;
      `Decompose(`r_1`,` P`,` T_{shape}`);`
      `Decompose(`r_2`,` P`,` T_{shape}`);`

B. RangeSearch on the Tree Tier (Algorithm 4)

Algorithm 4 RangeSearch

function RANGESEARCH(query point q, query range r, indoor index $\mathcal{T}$)
  set of objects R_o; set of partitions R_p; queue Q;
  Q.push($\mathcal{T}.root$);
  while Q is not empty **do**
    $t \leftarrow Q$.pop();
    if t is leaf node **then**
      if $|q,t|_{minK} \leq r$ **then**
        $R_p = R_p \cup t$;
        for each object O in t **do**
          if $|q,O|_{minK} \leq r$ **then** $R_o = R_o \cup O$;
    else
      for each child e of t **do**
        if $|q,e|_{minK} \leq r$ **then** Q.push(e);
  return R_o and R_p;

C. kSeedsSelection for ikNNQ (Algorithm 5)

Algorithm 5 kSeedsSelection

function KSEEDSSELECTION(query point q, k)
  set of objects R_o; set of partitions R_p; min-heap $\mathcal{H}$;
  find the Partition P_s containing q;
  push_heap($\mathcal{H}$, P_s);
  while $\mathbb{H}$ is not empty and $|R_o| < k$ **do**
    Add P to R_p;
    for each adjacent Partition P_i of P **do**
      push_heap($\mathcal{H}$, P_i);
      for each object O_j in P_i **do**
        Add O_j to R_o;
  Return R_o and R_p;