Indoor Spatial Queries: Modeling, Indexing, and Processing

Tiantian Liu[†] Huan Li[†] Hua Lu[‡] Muhammad Aamir Cheema[§] Lidan Shou[¶]
[†]Department of Computer Science, Aalborg University, Denmark
[‡]Department of People and Technology, Roskilde University, Denmark
[§]Faculty of Information Technology, Monash University, Australia
[¶]College of Computer Science, Zhejiang University, China
{liutt,lihuan}@cs.aau.dk, luhua@ruc.dk, aamir.cheema@monash.edu, should@zju.edu.cn

ABSTRACT

To support indoor spatial queries and indoor location-based services (LBS), multiple techniques including model/indexes and search algorithms have been proposed. In this work, we conduct an extensive experimental study on existing proposals for indoor spatial queries. We survey five model/indexes, compare their algorithmic characteristics, and analyze their space and time complexities. We also design an in-depth benchmark with real and synthetic datasets, evaluation tasks and performance metrics. Enabled by the benchmark, we obtain and report the performance results of all model/indexes under investigation. By analyzing the results, we summarize the pros and cons of all techniques and suggest the best choice for typical scenarios.

1 INTRODUCTION

Indoor location-based services (LBS) are becoming increasingly popular [6, 9]. Relevant applications, such as POI search [22, 28] and routing [11, 13, 14], are often built on top of typical spatial queries like range query, k nearest neighbor query, shortest path query, and shortest distance query. Therefore, the efficiency of processing such typical indoor spatial queries plays a key role in the success of indoor LBS.

To facilitate query processing for indoor LBS, space models, indexes and algorithms have been proposed. They all deal with indoor entities, e.g., rooms, doors, walls and floors. These entities form distinct topology that determines indoor distances and impacts indoor movement. As a result, the distances in indoor spatial queries must be measured appropriately, e.g., without involving straight line segments through walls. Also, indoor routing in shortest path/distance queries must consider connectivity and reachability between indoor locations.

To support indoor distance computation, existing models and indexes [27, 31, 37, 38] employ different approaches to integrate the geometry and topology information of an indoor space. Though all these approaches can be used to process the aforementioned indoor spatial queries, a comprehensive experimental study on all these proposals is still missing. Consequently, indoor LBS application developers inevitably encounter difficulties in choosing the appropriate technique for a given indoor space scenario.

To bridge this gap for LBS application development and disclose insights for further research on indoor data management, we conduct a comprehensive experimental study in this work. Our study focuses on five existing model/indexes that support typical indoor spatial queries on static indoor objects (e.g., POIs) or indoor shortest paths/distances. We compare the five proposals theoretically and empirically. Our contributions are as follows.

- We survey the five proposals by scrutinizing their structures, algorithmic characteristics, and space and time complexities.
- We design an in-depth benchmark with datasets, evaluation tasks, and performance metrics. The datasets consist of real and synthetic data characterized by distinctive indoor topology.
- Within the benchmark, we conduct extensive experiments to evaluate the performance of the five proposals in terms of construction cost and query efficiency.
- By analyzing the results, we disclose the pros and cons of the proposals, analyze the impact of different conditions, and recommend the best choice for typical application scenarios.

All code, data and test cases are open-sourced [1]. To the best of our knowledge, this work is the first that comparatively analyzes and evaluates the existing techniques under a unified framework.

The paper is organized as follows. Section 2 introduces indoor spatial queries and related work. Sections 3 and 4 present the indoor space model/indexes and query processing, respectively. Section 5 details the experimentation benchmark. Section 6 reports and analyzes the evaluation results. Section 7 concludes the paper.

2 INDOOR SPATIAL QUERIES

Table 1 lists the frequently used notations.

Table 1: Notations

Symbol	Meaning
$\mathbb{I}$	An indoor space
$p, q \in \mathbb{I}$	Indoor points
$o \in O$	A static indoor object
$d \in D$	A door
$v \in V$	An indoor partition
$\|p, q\|_I$	Indoor distance from p to q
$\langle p, d_i, \ldots, d_j, q\rangle$	An indoor path
$L(\phi)$	Length of a path ϕ

2.1 Indoor Space Concepts

Indoor space features distinct entities such as walls, doors, and rooms, which altogether form complex indoor topology that enables and constrains movements. Naturally, an indoor space is divided by walls and doors into **indoor partitions** like rooms, hallways or staircases. Two indoor partitions can be connected by a door or an open segment between them. Referring to the example floorplan in Figure 1, partitions 30 and 40 (denoted as v_{30} and v_{40}, respectively) are connected by an open segment d_3, In this paper, we refer to both doors and open segments as doors. We do not consider the width of a door and represent a door by its center point. In other words, each door can be generally regarded as an indoor point. Furthermore, a door can be unidirectional such as a security checkpoint at the airport. The door directionality makes the indoor distance between two points asymmetric. Referring to Figure 1, the shortest indoor path from p to p' and that from p' to p are different due to the unidirectionality of d_{12}.

Topology renders the indoor distance more complex than Euclidean distance. In Figure 1, the indoor distance $|p, o_1|_I$ from p to

o_1 is not subject to the straight line segment between them; it is the total length of the polyline $p \rightarrow d_{11} \rightarrow o_1$.

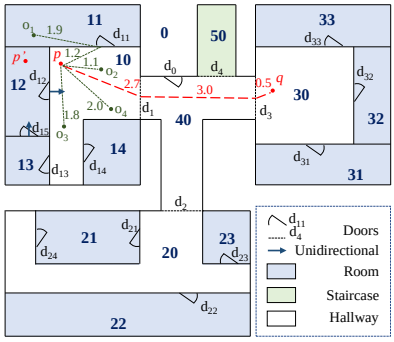


Figure 1: Example Floorplan

Lu et al. [27] proposes mappings to capture the relationships between indoor partitions and doors. In particular, $D2P_{\sqsupset}(d_i)$ gives the set of partitions that one can enter through door d_i and $D2P_{\sqsubset}(d_j)$ gives those that one can leave through door d_j. Besides, $D2P(d_i)$ gives a set of a partition pair (v_j, v_k) such that one can go through door d_i from partition v_j to v_k. Moreover, $P2D_{\sqsupset}(v_k)$ gives the set of enterable doors through which one can enter partition v_k, and $P2D_{\sqsubset}(v_k)$ gives the set of leaveable doors through which one can leave partition v_k. When doors are bidirectional, we use $P2D(v_k) = P2D_{\sqsupset}(v_k) \cup P2D_{\sqsubset}(v_k)$ to denote the set of doors associated to partition v_k.

Example 1. *In Figure 1, given the unidirectional door d_{12}, we have $D2P_{\sqsupset}(d_{12}) = \{v_{10}\}$, $D2P_{\sqsubset}(d_{12}) = \{v_{12}\}$, and $D2P(d_{12}) = \{(v_{12}, v_{10})\}$. Moreover, we have $P2D_{\sqsupset}(v_{12}) = \{d_{15}\}$, $P2D_{\sqsubset}(v_{12}) = \{d_{12}\}$, and $P2D(v_{12}) = \{d_{15}, d_{12}\}$.*

2.2 Indoor Spatial Query Types

We focus on *static* indoor objects such as POIs and facilities. Our study covers four fundamental indoor spatial query types.

Definition 1 (Range Query (RQ)). *Given an indoor point $p \in \mathbb{I}$, a set O of indoor objects, and a distance value r, a* range query *$\mathsf{RQ}(p, r)$ returns all indoor objects from O whose indoor distance from p is within r. Formally, $\mathsf{RQ}(p, r) = \{o \mid |p, o|_I \leq r, o \in O\}$.*

Definition 2 (k Nearest Neighbor Query (kNNQ)). *Given an indoor point $p \in \mathbb{I}$, a set O of indoor objects, and an integer value k, a k* nearest neighbor query *$k\mathsf{NNQ}(p)$ returns a set O' of k indoor objects whose indoor distances from p are the smallest, i.e., $|O'| = k$ and $\forall o_i \in O', o_j \in O \setminus O', |p, o_i|_I \leq |p, o_j|_I$.*

In Figure 1 where $O = \{o_1, \ldots, o_4\}$, a query $\mathsf{RQ}(p, 1.9\text{m})$ returns $\{o_2, o_3\}$ since the distances from p to o_1 and o_4 both exceed 1.9m.[1] Furthermore, a query $3\mathsf{NNQ}(p)$ returns $\{o_2, o_3, o_4\}$, since o_1's distance from p is the longest among all.

Definition 3 (Shortest Path Query (SPQ)). *Given a source point $p \in \mathbb{I}$, a target point $q \in \mathbb{I}$, a* shortest path query *$\mathsf{SPQ}(p, q)$ returns the shortest path $\phi = \langle p, d_i, \ldots, d_j, q \rangle$ from p to q such that 1) $d_i, \ldots, d_j$ are door sequences and each two consecutive doors are associated to the same partition, 2) p is in the partition having d_i as a leavable door, 3) q is in the partition having d_j as an enterable door, and 4) $\forall \phi'$ from p to q, $L(\phi) \leq L(\phi')$.*[2]

Definition 4 (Shortest Distance Query (SDQ)). *Given a source point $p \in \mathbb{I}$, a target point $q \in \mathbb{I}$, a* shortest distance query *$\mathsf{SDQ}(p, q)$ returns the shortest indoor distance from p to q, i.e., the length of $\mathsf{SPQ}(p, q)$.*

[1] Meter is the distance unit in all examples in this paper.

[2] $L(\phi) = \sum_{k=0}^{k=j} |d_k, d_{k+1}|_I$ where $d_0 = p$ and $d_{j+1} = q$.

As indicated by the red dashed polyline in Figure 1, a query $\mathsf{SPQ}(p, q)$ returns $\phi = \langle p, d_1, d_3, q \rangle$ as the shortest path from p to q, and the result of $\mathsf{SDQ}(p, q)$ is 2.7m + 3.0m + 0.5m = 6.2m.

2.3 Related Work

Indoor Space Modeling. Many indoor space models [7, 15, 20, 35, 36] focus on symbolic modeling of topological relationships between indoor partitions. Lacking of indoor distances, they cannot support the aforementioned distance-aware queries.

Indoor Moving Objects. Alamri et al. [4] propose an index tree for indoor moving objects based on connectivity between indoor cellular units. Kim et al. [19] propose to index indoor moving objects based on grid cells. Lin et al. [24] design an indoor moving object index to speed up complex semantic queries in multi-floor spaces. In the context of RFID indoor tracking, Yang et al. study continuous range monitoring queries [39] and probabilistic k nearest neighbor queries [40]. To improve the query result, Yu et al. [41] propose a particle filter-based method to infer the undetected locations of indoor moving objects. Assuming a probabilistic sample based location data format, Xie et al. [37, 38] process kNN query and range query for indoor moving objects. Considering uncertain object movements between observed time and query time, Li et al. [22] study searching the current top-k indoor dense regions. These works consider indoor moving objects with uncertain positions at a particular time. Unlike all these works on indoor moving objects, this study concerns spatial queries on static indoor objects, e.g., printers or ATMs.

Indoor Trajectories. Jensen et al. [15] study historical trajectories of RFID-tracked indoor objects. Delafontaine et al. [12] find sequential visiting patterns within historical Bluetooth tracking data. Given a past time or a time interval, Lu et al. define spatio-temporal joins [29] to find moving object pairs in the same indoor partition, and top-k queries [28] to find the most frequently visited indoor POIs. Ahmed et al. [2, 3] define threshold density query to find dense indoor semantic locations in a historical time interval. Assuming probabilistic sample based location records, Li et al. [23] find the top-k most popular indoor semantic regions with the highest object flow values. Jin et al. [17] study the similarity search over indoor trajectories, considering both spatial and semantic properties. By analyzing spatial constraints of indoor POIs, Jiang et al. [16] study the restoration of indoor trajectories. Li et al. [21] propose a coupled conditional Markov model to enrich indoor uncertain trajectories with mobility events and stay regions. Unlike these works, the queries studied in this paper focus on static objects or indoor paths.

Indoor Path Planning. Goetz and Zipf [14] study user-adaptive length-optimal indoor routing based on a weighted routing graph. Salgado et al. [30] study indoor keyword-aware skyline route query, considering the number of covered keywords and route distances. Feng et al. [13] study indoor keyword-aware routing queries to find shortest paths covering user-specified semantic keywords. Costa et al. [11] propose the context-aware indoor-outdoor path recommendation that minimizes the outdoor exposure and path distance. To enable navigation through movable obstacles, Sun et al. [33] study semantic assisted path planning over a gridded map of an indoor environment. Wang et al. [34] propose an obstacle-avoiding path planning algorithm to automate indoor robots. These techniques consider additional query semantics, and thus are different from the fundamental, pure shortest path/distance queries studied in this paper.

3 MODEL AND INDEXES

The aforementioned indoor spatial queries all involve indoor distances. To facilitate such queries, indoor distances must be considered in modeling and indexing indoor space.

3.1 Indoor Distance-Aware Model

Indoor distance-aware model [27] (IDMODEL) is a graph $G_{\text{dist}}(V, E_a, L, f_{\text{dv}}, f_{\text{d2d}})$. The first three elements capture indoor topology in an *accessibility base graph* $G_{\text{accs}}(V, E_a, L)$, where V is the set of vertexes each referring to an indoor partition, $E_a = \{(v_i, v_j, d_k) \mid d_k \in D, v_i \in D2P_{\sqsupset}(d_k) \wedge v_j \in D2P_{\sqsubset}(d_k)\}$ is a set of labeled, directed edges, and L is the set of edge labels each corresponding to a door in D. The additional two are mapping functions defined as follows.

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

Here, $||p, q||_{v_j}$ is the indoor distance from a point p to a point q within the partition v_j. Note that $||p, q||_{v_j}$ is not necessarily a Euclidean distance because even within the same partition there may be obstacles in the line of sight between p and q. Specifically, *door-to-partition distance mapping* $f_{\text{dv}}(d_i, v_j)$ returns the longest distance one can reach within partition v_j from door d_i, if v_j is an enterable partition of d_i. Otherwise, it returns ∞.

$$f_{\text{d2d}}(v_j, d_i, d_j) = \begin{cases} ||d_i, d_j||_{v_j}, & \text{if } d_i \in P2D_{\sqsupset}(v_j) \\ & \text{and } d_j \in P2D_{\sqsubset}(v_j); \\ 0, & \text{if } d_i = d_j \\ & \text{and } d_i, d_j \in P2D(v_j); \\ \infty, & \text{otherwise.} \end{cases}$$

The *door-to-door distance mapping* $f_{\text{d2d}}(v_j, d_i, d_j)$ maps a partition v_j and two doors d_i and d_j to a distance value. If both doors are associated to v_j, it returns the distance from d_i to d_j within v_j, i.e., $||d_i, d_j||_{v_j}$. If d_i and d_j are identical and associated to v_j, we stipulate $f_{\text{d2d}}(v_j, d_i, d_j) = 0$. Otherwise, $f_{\text{d2d}}(v_j, d_i, d_j)$ returns ∞, indicating that one cannot go from d_i to d_j via v_j only.

Figure 2 illustrates the IDMODEL for the example shown in Figure 1. The outdoor space is captured in a special graph vertex v_0. Two hashmaps implement the mappings $f_{\text{dv}}(d_i, v_j)$ and $f_{\text{d2d}}(v_j, d_i, d_j)$. With directed edges, IDMODEL can support doors' directionality and temporal variation when needed.

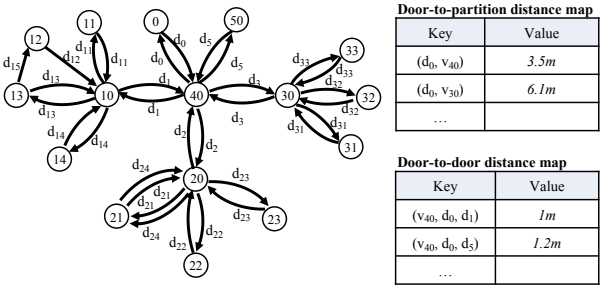


Figure 2: An Example of IDMODEL

With the two mappings $f_{\text{dv}}(d_i, v_j)$ and $f_{\text{d2d}}(v_k, d_i, d_j)$, a graph traversal algorithm [27] on IDMODEL is designed to compute the shortest door-to-door distance $\text{d2d}(d_s, d_t)$ from a source door d_s to a target door d_t. The basic idea is to keep expanding to unvisited doors based on the current shortest path until reaching the target door. Further, the shortest indoor distance from any point p to any point q can be computed by finding the minimum value of the distance summation $||p, d_p||_{v_p} + \text{d2d}(d_p, d_q) + ||d_q, q||_{v_q}$, where v_p and v_q are the partitions that host p and q, respectively, $d_p \in P2D_{\sqsubset}(v_p)$, and $d_q \in P2D_{\sqsupset}(v_q)$.

However, IDMODEL does not support fast determination of the host partition of a query/source point. It boils down to sequential scanning of all partitions if no additional index, e.g., R-tree, is used for the partitions. Also, to manage indoor static objects, IDMODEL needs additional object buckets each for a partition.

3.2 Indoor Distance-Aware Index

IDMODEL only captures the door-to-door and door-to-partition distances within a local partition, which entails extra search to compute the indoor distance for two points in different partitions.

To cut such costs, indoor distance-aware index [27] (IDINDEX) stores extra information on top of IDMODEL, namely, precomputed global door-to-door distances and their ordering in two matrices. The **door-to-door distance matrix** M_{d2d} is an N-by-N matrix where $\text{N} = |D|$ is the total number of doors and $M_{\text{d2d}}[d_i, d_j]$ gives the precomputed shortest indoor distance from d_i to d_j. The **distance index matrix** M_{idx} is also an N-by-N matrix such that $M_{\text{idx}}[d_i, k]$ gives the identifier of a door whose indoor distance from d_i is the k-th shortest among all the N doors.

The IDINDEX matrices for the top-left part in Figure 1 is illustrated in Figure 3. Here, we have $M_{\text{d2d}}[d_1, d_{15}] = 4.6\text{m}$. The first row of M_{d2d} shows that d_{15} has the longest indoor distance from d_1. Accordingly, we have $M_{\text{idx}}[d_1, 6] = d_{15}$ in M_{idx}.

$$\begin{pmatrix} & d_1 & d_{11} & d_{12} & d_{13} & d_{14} & d_{15} \\ d_1 & 0 & 1.7 & 2.7 & 3.6 & 2.8 & 4.6 \\ d_{11} & 1.7 & 0 & 1.9 & 3.6 & 2.8 & 4.6 \\ d_{12} & 2.7 & 1.9 & 0 & 2.6 & 1.8 & 1.6 \\ d_{13} & 3.2 & 3.4 & 2 & 0 & 2 & 1 \\ d_{14} & 2.8 & 2.8 & 1.8 & 1 & 0 & 2 \\ d_{15} & 4.3 & 3.5 & 1.6 & 1 & 2 & 0 \end{pmatrix} \qquad \begin{pmatrix} & 1 & 2 & 3 & 4 & 5 & 6 \\ d_1 & d_1 & d_{11} & d_{12} & d_{14} & d_{13} & d_{15} \\ d_{11} & d_{11} & d_1 & d_{12} & d_{14} & d_{13} & d_{15} \\ d_{12} & d_{12} & d_{15} & d_{14} & d_{11} & d_{13} & d_1 \\ d_{13} & d_{13} & d_{15} & d_{12} & d_{14} & d_1 & d_{11} \\ d_{14} & d_{14} & d_{13} & d_{12} & d_{15} & d_1 & d_{11} \\ d_{15} & d_{15} & d_{13} & d_{12} & d_{14} & d_{11} & d_1 \end{pmatrix}$$

(a) Distance Matrix M_{d2d} (b) Distance Index Matrix M_{idx}

Figure 3: An Example of IDINDEX

As the shortest indoor distances to all doors are precomputed and sorted for each door in IDINDEX, it is faster to compute the shortest indoor distance between any two points p and q in the indoor space. To support the shortest path query, in addition to the shortest distance value between any two points, IDINDEX also keeps the first-hop door of the corresponding shortest path. In this way, the complete shortest path between two points can be constructed by recursively concatenating the first-hop doors.

3.3 Composite Indoor Index

Composite indoor index [37] (CINDEX) is a layered structure for indexing indoor partitions and moving objects. It consists of three layers: geometric layer, topological layer, and object layer. In this study, we adapt the object layer to index static indoor objects. A partial example CINDEX for Figure 1 is given in Figure 4.

The **geometric layer** uses an R*-tree [8] to index all indoor partitions, with an additional skeleton tier to maintain the distances between staircases at different floors. To ease the geometrical computations, it decomposes each irregular partition[3] into regular ones using a decomposition algorithm [37]. Referring to the bottom-right of Figure 4, the hallway v_{10} is divided into two regular indoor partitions v_{10a} and v_{10b} by a door d_{16}. Afterwards, each regular partition is represented by a *Minimum Bounding Rectangle* (MBR). The MBRs are indexed by the R*-tree. As shown in the top-left of Figure 4, a non-leaf node R_1 is composed of six partitions in the leaf level, i.e., v_{10a}, v_{10b}, and v_{11}-v_{14}.

[3] A partition is irregular if it is non-convex or imbalanced (long in one dimension but short in the other).

The **topological layer** stores the connectivity information among indoor partitions, and it is integrated to the tree by inter-partition links. In particular, a leaf node v_i in the R*-tree is linked with a pointer record $(d_k, \uparrow v_j)$ to indicate that one can move from a partition v_i to another partition v_j through door d_k. As shown in the top-right of Figure 4, the two pointer records for v_{13} mean that v_{13} is adjacent to v_{10b} and v_{12} via d_{13} and d_{15}, respectively.

The **object layer** maintains a number of object buckets each for an indoor partition at the leaf node level of the R*-tree. Each indoor object o is kept in the bucket of the partition in which o is located. In addition, an object hashtable o-table : $O \rightarrow {}^*V$ maps each object to its host partition's pointer. Unlike [37, 38], the object buckets store static objects in this study. As shown in the bottom-left of Figure 4, the leaf node v_{10a} is linked to its object bucket with two static objects o_2 and o_4. Also, two corresponding records are kept in the object hashtable (o-table).

The R*-tree in CINDEX organizes partitions hierarchically, and thus enables search space pruning for distance relevant computations. As a result, CINDEX does not cache the precomputed door-to-door distances as IDINDEX does. Moreover, as the topological layer maintains the links between partitions and doors, which form an implicit graph structure, CINDEX does not need an explicit graph model to keep connectivity information. The topological layer's dynamic link updating makes CINDEX adaptive to possible temporal variations of doors.

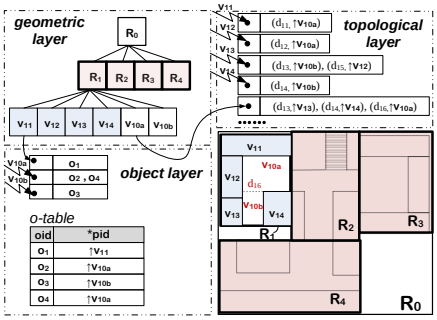


Figure 4: CINDEX Example (Adapted from [37])

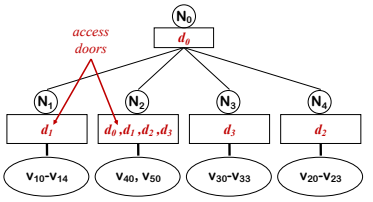
3.4 IP-Tree and VIP-Tree

Indoor partitioning tree [31] (IP-TREE) is a tree-based indoor partition index with a number of matrices each materializing the door-to-door distances within a local range. In particular, each leaf node of IP-TREE covers a number of topologically adjacent indoor partitions. The adjacent leaf nodes are combined to form a non-leaf node, and adjacent non-leaf nodes are combined hierarchically until a root node is formed. Each node N has a **distance matrix** and a number of **access doors**. An access door is a border door that connects N to its external space. AD(N) denotes N's access door set. The distance matrix for a leaf node stores the shortest distance (as well as the first-hop door on the shortest path) between every door of the leaf node to every access door of the leaf node. The distance matrix for a non-leaf node only stores the shortest distances and first-hop door between each pair of access doors of its child nodes. To compute the indoor distance from a point p to a point q, IP-TREE locates the lowest common ancestor of the leaf nodes $\texttt{Leaf}(p)$ and $\texttt{Leaf}(q)$, finds the access doors constituting the shortest path in that ancestor, and connects the materialized indoor distances involving p, the found access doors, and q.

Figure 5 shows an example of IP-TREE corresponding to Figure 1. The topologically adjacent partitions v_{10}-v_{14} form a leaf node N_1. Another leaf node N_2 is composed of partitions v_{40} and v_{50}. As N_1 and N_2 are connected by a border door d_1, d_1 is put into AD(N_1) and AD(N_2). For the leaf node N_1, the distance matrix stores the distances from each of its doors to the access door d_1 of N_1. For instance, the distance from N_1's only door d_{15} to access door d_1 contained by N_1 is 4.3m. Moreover, as the shortest path from d_{15} to d_1 is $\langle d_{15}, d_{12}, d_1 \rangle$, the first-hop door of the path is kept as d_{12} in the matrix. Differently, for the non-leaf node N_0, the distance matrix only keeps the distances between each pair of access doors. In the running example, each pair of access doors are directly connected. Therefore, no first-hop door is recorded. The storage space of each distance matrix will double when the door directionality needs to be considered, i.e., both the distances d2d(d_i, d_j) and d2d(d_j, d_i) are kept in each node.

As a variant of IP-TREE, vivid IP-Tree (VIP-TREE) [31] further accelerates the distance computation by materializing more precomputed information. Specifically, each leaf node N additionally maintains the shortest distance between each door contained by N and each access door in N's all ancestor nodes, along with the corresponding first-hop door information.

IP-TREE and VIP-TREE materialize a small number of distances only related to access doors that are critical in the overall topology of an indoor space. This design eases the on-the-fly distance related computations in spatial query processing.



	d_0	d_1	d_2	d_3
d_0	0	1.4	2	3.9
d_1	1.4	0	3	4
d_2	3.9	4	4.4	0
d_3	2	3	0	4.4

Distance Matrix for $\mathbf{N_0}$ (a non-leaf node)

	d_1	d_{11}	d_{12}	d_{13}	d_{14}	d_{15}
d_1	0	1.7	2.7	3.2	2.8	4.3, d_{12}

Distance Matrix for $\mathbf{N_1}$ (a leaf node)

Figure 5: An Example of IP-TREE

4 QUERY PROCESSING

All the aforementioned model/indexes can be used to process indoor spatial queries. Although query processing differs for different query types, all algorithms share a general paradigm as follows. First, an algorithm finds the initial indoor partition for a query. The initialization decides the indoor partition in which the query (or source) point p is located for a given RQ(p, r) (kNNQ(p), SPQ(p, q), or SDQ(p, q)). Subsequently, an algorithm expands from the initial partition, searching adjacent partitions via doors. Finally, the expansion stops when the search range is beyond the query range r for a RQ(p, r), or kNNs have been found for a kNNQ(p), or the target point q is met for a SPQ(p, q) or SDQ(p, q). Algorithms based on different model/indexes differ in their initializations and expansions. Below, we present a comprehensive analytical comparison of all model/indexes.

4.1 Algorithmic Comparison

Table 2 summarizes the comparison.

Distance Precomputation. IDMODEL and CINDEX do not precompute any indoor distances, whereas IDINDEX and IP-TREE/VIP-TREE maintain some door-to-door distances before query processing. In particular, IDINDEX precomputes the shortest indoor distances between every pair of doors, but IP-TREE/VIP-TREE only keeps a small number of distances in each tree node.

Model/Index Structure. IDMODEL is a labeled graph with distance mapping functions, whereas IDINDEX materializes two matrices for global door-to-door distances. Employing a tree-based structure, CINDEX keeps topological information incrementally by maintaining inter-partition links, whereas IP-TREE/VIP-TREE augments each tree node with a local distance matrix. More importantly, CINDEX forms the non-leaf tree nodes according to the

Table 2: Feature Comparison

Models	Precompute	Structure	Initialization	Expansion	RQ	kNNQ	SPQ	SDQ
IDModel	No	Graph+ Mappings	Sequential scan	Dijkstra	△	△	✓	✓
IDIndex	Yes	Matrix	Sequential scan	Loop	✓	✓	△	△
CIndex	No	Tree+Links	R*-Tree pruning	Dijkstra	✓	✓	△	△
IP-Tree	Yes	Tree+Matrix	Sequential scan	LCA	✓	✓	✓	✓
VIP-Tree	Yes	Tree+Matrix	Sequential scan	LCA	✓	✓	✓	✓

Table 3: Extensibility Analysis

	IDModel	IDIndex	CIndex	IP/VIP-Tree
Temporal Variation	✓	X	✓	X
Moving Objects	✓	✓	✓	✓
Uncertain Locations	X	X	✓	X
Keywords	✓	✓	✓	✓

Table 4: Complexity Analysis

	Space	RQ	kNNQ	SDQ	SPQ
IDModel	$O(\mathtt{V}+\mathtt{D}+2\mathtt{Vd}+\mathtt{Vd}^2)$	$O(\mathtt{oV}\log\mathtt{D})$	$O(\mathtt{oV}\log\mathtt{D})$	$O(\mathtt{V}\log\mathtt{D})$	$O(\mathtt{V}\log\mathtt{D}+\mathtt{w})$
IDIndex	$O(2\mathtt{D}^2)$	$O(\mathtt{od}\log\mathtt{D})$	$O(\mathtt{od}\log\mathtt{D})$	$O(\mathtt{d}^2)$	$O(\mathtt{d}^2+\mathtt{w})$
CIndex	$O(\mathtt{V}+\mathtt{Vd}+\mathtt{O})$	$O(\mathtt{oV}\log\mathtt{D})$	$O(\mathtt{oV}\log\mathtt{D})$	$O(\mathtt{V}\log\mathtt{D})$	$O(\mathtt{V}\log\mathtt{D}+\mathtt{w})$
IP-Tree	$O(\rho^2\mathtt{f}^2\mathtt{L}+\rho\mathtt{D})$	$O((\rho\log_\mathtt{f}\mathtt{L})^2(\mathtt{Vo}/\mathtt{L}+\rho))$	$O((\rho\log_\mathtt{f}\mathtt{L})^2(\mathtt{Vo}/\mathtt{L}+\rho))$	$O(\rho^2\log_\mathtt{f}\mathtt{L})$	$O((\rho^2+\mathtt{w})\log_\mathtt{f}\mathtt{L})$
VIP-Tree	$O(\rho^2\mathtt{f}^2\mathtt{L}+\rho\mathtt{D}\log_\mathtt{f}\mathtt{L})$	$O(\rho^2\log_\mathtt{f}\mathtt{L}(\mathtt{Vo}/\mathtt{L}+\rho))$	$O(\rho^2\log_\mathtt{f}\mathtt{L}(\mathtt{Vo}/\mathtt{L}+\rho))$	$O(\rho^2)$	$O(\rho^2+\mathtt{w})$

geometrical proximity of partitions, whereas IP-Tree/VIP-Tree do so based on the topological proximity of partitions.

Query Types. All model/indexes can support all the four query types. However, IDModel [27] does not provide RQ and kNNQ algorithms. Therefore, we implement the two algorithms and refer readers to the appendix in [26]. Also, there are no off-the-shelf SPQ/SDQ algorithms for IDIndex and CIndex. Nevertheless, the global door-to-door distances and the corresponding last-hop door information in IDModel can be used to expand path searching in SPQ/SDQ algorithms for IDIndex. For CIndex, the inter-partition links can be used for path expansion.

Initialization. To decide the initial indoor partition for a query, IDModel and IDIndex sequentially scan all partitions. Enabled by the R*-tree indexing partitions, CIndex can quickly find the host partition of any indoor point. In contrast, IP-Tree and VIP-Tree are based on pure topological relationships among partitions, and thus they also sequentially scan all partitions.

Expansion. As a graph-based model, IDModel expands to the next unvisited door in the spirit of Dijkstra's algorithm [18]. CIndex does so as well since the next-hop doors are captured in the inter-partition links on the topological layer. Instead of expanding via directly connected doors, IP-Tree/VIP-Tree finds the lowest common ancestor (LCA) node of p and q and locates the intermediate access doors on the shortest path straightforwardly. It is noteworthy that IDIndex alone cannot support topological door expansion. Instead, IDIndex relies on an underlying IDModel to loop through relevant indoor partitions' doors.

4.2 Complexity Analysis

Let $\mathtt{V}$, $\mathtt{D}$, $\mathtt{O}$ be the total number of indoor partitions, doors, and indoor objects, respectively. Let $\mathtt{d}$ and $\mathtt{o}$ be the average door number and average object number per partition, respectively. Let $\mathtt{w}$ be the average number of door nodes on a shortest path. For IP-Tree/VIP-Tree, we use $\mathtt{f}$ to denote the fan-out of the tree node, ρ the average access door number per node, and $\mathtt{L}$ the total number of leaf nodes. Table 2 summarizes the space complexity of all model/indexes and their time complexity for queries.

Space Complexity. IDModel $(V, E_a, L, f_{\mathrm{dv}}, f_{\mathrm{d2d}})$'s space complexity is $O(\mathtt{V}+\mathtt{Vd}+\mathtt{D}+\mathtt{Vd}+\mathtt{Vd}^2)=O(\mathtt{Vd}^2)$. IDIndex's space complexity is $O(2\mathtt{D}^2)=O(\mathtt{D}^2)$ as it consists of two door matrices. CIndex's space complexity is $O(\mathtt{V}+\mathtt{Vd}+\mathtt{O})=O(\mathtt{Vd}+\mathtt{O})$ where $\mathtt{V}$, $\mathtt{Vd}$, and $\mathtt{O}$ correspond to partition R*-tree, inter-partition links, and object hashtable, respectively. IP-Tree's space cost mainly consists of the distance matrices for leaf nodes and those for non-leaf nodes. The former's complexity is $O(\rho\mathtt{D})$ and the latter's is $O((\rho\mathtt{f})^2\mathtt{L})$ where $\rho\mathtt{f}$ corresponds to the number of access doors from a child node and $\mathtt{L}$ reflects the number of non-leaf nodes. In contrast, VIP-Tree's space cost on the distance matrices for leaf nodes is $O(\rho\mathtt{D}\log_\mathtt{f}\mathtt{L})$, where $\log_\mathtt{f}\mathtt{L}$ corresponds to the ancestor number of each leaf node.

Time Complexity for RQ and kNNQ. RQ and kNNQ have similar time complexity as they both prune objects based on shortest distances. IDModel's search expands via qualified doors by graph traversal in $O(\mathtt{V}\log\mathtt{D})$ and iterates on the objects in each visited partition in $O(\mathtt{o})$. Also based on graph traversal, the search on CIndex obtains a subgraph in $O(\mathtt{V}\log\mathtt{D})$ and visits all objects in each partition of the subgraph in $O(\mathtt{o})$. IDIndex's search expands to the nearest partitions based on the sorted result in M_{idx}, and loops through each object in the expanded partition. So its time complexity is $O(\mathtt{od}\log\mathtt{D})$. The searches via IP-Tree and VIP-Tree work similarly. They prune a tree node based on its distance from the query point in $O(\log_\mathtt{f}\mathtt{L}\cdot\rho\cdot c)$, where c is the unit SDQ cost. Then, they qualify each object in the remaining nodes in $O(\log_\mathtt{f}\mathtt{L}\cdot\mathtt{V}/\mathtt{L}\cdot\mathtt{o}\cdot c)$. Given the SDQ complexity $O(\rho^2\log_\mathtt{f}\mathtt{L})$ for IP-Tree and $O(\rho^2)$ for VIP-Tree (to be detailed below), their RQ and kNNQ complexities are $O((\rho\log_\mathtt{f}\mathtt{L})^2(\mathtt{Vo}/\mathtt{L}+\rho))$ and $O(\rho^2\log_\mathtt{f}\mathtt{L}(\mathtt{Vo}/\mathtt{L}+\rho))$, respectively.

Time Complexity for SDQ and SPQ. For the graph traversal algorithms of IDModel and CIndex, the SDQ complexity is $O(\mathtt{V}\log\mathtt{D})$ and SPQ complexity is $O(\mathtt{V}\log\mathtt{D}+\mathtt{w})$ with additional cost to backtrack the shortest path in $\mathtt{w}$ hops. For IDIndex, the only cost of SDQ is to loop through two door sets corresponding to p and q by a complexity of $O(\mathtt{d}^2)$. The extra cost of SPQ to concatenate shortest path is of $O(\mathtt{w})$. For IP-Tree, SDQ needs to search the lowest common ancestor and then find a pair of access doors from that ancestor node, resulting in a complexity of $O(\rho^2\log_\mathtt{f}\mathtt{L})$. In contrast, VIP-Tree materializes the distances from a leaf node to each access door in the ancestors. Its SDQ complexity is $O(\rho^2)$. The additional cost to construct shortest path in SPQ is $O(\mathtt{w}\log_\mathtt{f}\mathtt{L})$ for IP-Tree and $O(\mathtt{w})$ for VIP-Tree.

4.3 Extensibility Analysis

Table 3 summarizes the extensibility of all model/indexes.

Temporal Variation. Indoor topology may feature temporal variations, e.g., doors have open and close hours. To support indoor spatial queries in such cases, temporal variations like open and close time of doors can be maintained as a table attached to the accessibility base graph of IDModel or the topological layer of CIndex [25]. However, frequent temporal variations are hard to handle for IDIndex and IP-Tree/VIP-Tree as they need to precompute door-to-door distances globally or locally.

Moving Objects. CIndex [37, 38] is designed for managing indoor moving objects. It supports distance-aware queries like kNNQ and RQ, and also distance-aware joins like semi-range join and semi-neighborhood join. All other model/indexes can also index moving objects by maintaining dynamic object buckets

attached to indoor partitions in a way similar to how we handle the static objects. Nevertheless, the buckets need to be updated appropriately for indoor moving objects.

Uncertain Locations. In some settings, indoor points or objects are represented as uncertain regions. To process indoor spatial queries over uncertain locations, a model/index should support geometric operations on partitions. As a result, only CINDEX with partition R*-tree excels at handling uncertain locations [37, 38].

Keywords. A spatial keyword query [10] returns objects or paths that are spatially and textually relevant to the user-specified location(s) and keyword(s). Such queries can be supported if we extend the model/indexes by additionally maintaining mappings between partitions/objects and keywords. Especially, top-k keyword-aware shortest path queries have been supported based on IDMODEL [13], and boolean kNN spatial keyword queries have been supported based on VIP-TREE [32].

5 BENCHMARK

In this section, we detail the benchmark for evaluating the indoor spatial query techniques (model/indexes and algorithms). All code, data, and test cases are available online [1].

5.1 Datasets

We use four very different indoor space datasets, each featuring a distinctive indoor topology. The floorplans are briefly represented and illustrated in Figure 6. The data statistics are given in Table 5.

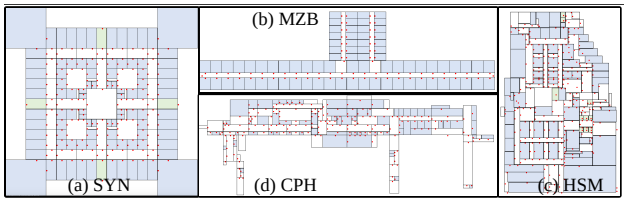


Figure 6: Floorplan of Datasets.

Synthetic Building (**SYN**) is a n-floor building. Its each floor is from a real-world floorplan [4] of 1368m × 1368m with 141 partitions and 216 doors. Its each two adjacent floors are connected by four 20m long stairways. By default, we set $n = 5$ and get the default dataset SYN5. To study the effect of topological changes, from SYN5 we derived SYN5$^-$ with fewer doors and SYN5$^+$ with more doors. Note that varying the door number will significantly change the connectivity and accessibility of the partitions, leading to a major topological change. We also form SYN5^0 in which the hallways are not decomposed [5].

Menzies Building (**MZB**) [6] is a landmark building at Clayton campus of Monash University. Each floor takes approximately 125m × 35m and connects to adjacent floors by two or four stairways each being 5m long. In total, there are 1344 partitions (including 34 staircases and 85 hallways) and 1375 doors. By changing the hallway decomposition, we form MZB^0 in which the hallways are not decomposed and MZB$^\Delta$ in which the hallways are decomposed into more partitions than default.

Hangzhou Shopping Mall (**HSM**) is a 7-floor mall in Hangzhou, China, occupying 2700m × 2000m. Ten stairways connect each two adjacent floors. Each floor contains 150 partitions and 299 doors on average. In total, there are 1050 partitions (including 70 staircases and 133 hallways) and 2093 doors.

[4] https://deviantart.com/mjponso/art/Floor-Plan-for-a-Shopping-Mall-86396406

[5] We precompute the door-to-door distance matrix for each hallway when it is not decomposed. The hallways are of irregular and concave shapes, and thus the door-to-door distance in a hallway can not use the Euclidean distance.

[6] https://www.monash.edu/virtual-tours/menzies-building

Copenhagen Airport (**CPH**) refers to the ground floor of Copenhagen Airport [7], taking around 2000m × 600m with 147 partitions (including 25 hallways) and 211 doors.

Overall Analysis of Different Datasets. The statistics of the datasets are given in Table 5. We use $\#dv$ to denote the number of doors in a partition, and conduct quartile statistics [5] on $\#dv$. In Table 5, Q1($\#dv$), Q2($\#dv$), and Q3($\#dv$) denote the first, second, third quartiles of $\#dv$, respectively, and max($\#dv$) denotes the maximum value of $\#dv$. In addition, we also plot the distributions of $\#dv$ over all partitions in each dataset in Figure 7.

Table 5: Statistics of Datasets

Datasets	SYN	MZB	HZM	CPH	SYN5$^-$	SYN5$^+$	SYN5^0	MZB^0	MZB$^\Delta$
Floors	n	17	7	1	5	5	5	17	17
Doors	216n	1375	2093	211	840	1280	880	1308	1480
Partitions	141n	1344	1050	147	705	705	505	1276	1449
Hallways	41n	85	483	72	205	205	5	17	190
C-Pars	8n	52	133	20	20	40	5	19	157
Length(m)	1368	125	2700	2000	1368	1368	1368	125	125
Width(m)	1368	35	2000	600	1368	1368	1368	35	35
Q1($\#dv$)	2	1	2	1	1	2	1	1	1
Q2($\#dv$)	2	1	4	2	1	3	2	1	1
Q3($\#dv$)	4	1	5	4	3	4	3	1	1
max($\#dv$)	10	56	17	12	10	10	132	82	47

Based on the space scale information and door distribution information from Table 5 and Figure 7, we summarize the characteristics of each dataset as follows.

- SYN: The overall space is square and regular. The number of doors and partitions in each floor is medium (216 doors and 141 partitions per floor). The door density within each partition is small (with Q2 equals only 2).
- MZB: The overall space is long and narrow with large scale crucial partitions (*C-Pars* for short). The number of doors and partitions in each floor is relatively small (80.4 doors and 76.8 partitions on average), whereas the overall size of doors and partitions is large due to the floor number. The planning of doors is rather skewed in that most partitions have only 1 or 2 doors while there are some C-Pars that accommodate 56 doors (as shown in Figure 7(b)).
- HSM: The overall space is long and relatively narrow. The number of doors and partitions in each floor is medium and the overall size of doors and partitions is large. The planning of doors is regular and door density in each partition is medium (Q2 and Q3 are equal to 4 and 5, respectively).
- CPH: The space is long, narrow yet open, resulting in a small number of doors and partitions. The door distribution is regular and door density in each partition is small (Q2 equals 2).

5.2 Object/Query Workload Generation

For each dataset, we randomly generated a set O of valid points as static objects, each object in O falling in an indoor partition. To test the effect of different object numbers, we vary $|O|$ as 500, 1000, 1500, 2000 and 2500.

The augment generation for each query type is detailed below.

RQ(p, r). We vary the range value r according to the predefined values in Table 6 (default values in bold). For each r, we generate ten RQ instances with a random p in the indoor space.

kNNQ(p). Similar to RQ generation, we generate ten random kNNQ instances for each k value given in Table 6.

As SPQ and SDQ can be integrated into one search procedure, we use SPDQ(p, q) to denote the integrated query that returns the shortest path from p to q along with the corresponding shortest distance value. In the following sections, we evaluate search performance of SPDQ only.

[7] https://www.cph.dk/en/practical

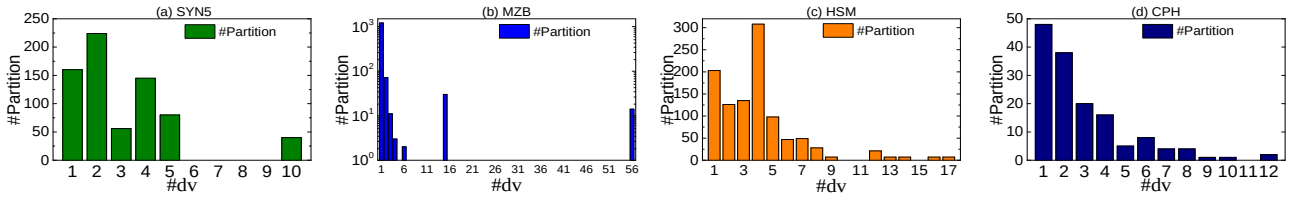


Figure 7: Distribution of $\#dv$ (number of doors in a partition) on (a) SYN5, (b) MZB, (c) HSM, and (d) CPH.

Table 6: Evaluation Settings (Default Parameters in Bold)

Symbol & Meaning		Task	Metrics	Queries	Dataset	Parameter Setting
n	floor number	A B1	a1, a2 b1, b2, b3 (only for SPDQ)	- RQ, kNNQ, SPDQ	SYN	3, **5**, 7, 9
$\|O\|$	object number	B2	b1, b2	RQ, kNNQ	all	500, 1000, **1500**, 2000, 2500
r	range value	B3	b1, b2	RQ	SYN5, HZM, CPH MZB	200, 400, **600**, 800, 1000 20, 40, **60**, 80, 100
k	-	B4	b1, b2	kNNQ	all	1, 5, **10**, 50, 100
s2t	source-target distance	B5	b1, b2, b3	SPDQ	SYN5, HZM, CPH MZB	1100, 1300, **1500**, 1700, 1900 30, 60, **90**, 120, 150
-	topological change	B6	b1, b2, b3 (only for SPDQ)	RQ, kNNQ, SPDQ	SYN	SYN5$^-$, SYN5, SYN5$^+$
-	decomposition method	B7	b1, b2, b3 (only for SPDQ)	RQ, kNNQ, SPDQ	SYN MZB	SYN5^0, SYN5 MZB^0, MZB, MZB$^\Delta$

SPDQ(p, q). We use a parameter s2t to control the shortest distance from the source p and target q. Its parameter values are listed in Table 6. For each s2t, we generate ten different (p, q) pairs to form SPDQ instances as follows. First, we randomly select an indoor point p and find a door d whose indoor distance from p approximates s2t. Next, we expand from d to find a random point q whose indoor distance from p approximates s2t.

5.3 Model/Index Settings

IDMODEL. For each partition v_i, we implemented the door-to-door distance mapping $f_{\text{d2d}}(v_i, \cdot, \cdot)$ as a 2D array, and door-to-partition distance mapping $f_{\text{dv}}(\cdot, v_i)$ as an 1D array. Besides, the partition mappings $P2D_{\sqsupset}(v_i)$ and $P2D_{\sqsubset}(v_i)$ (cf. Section 2.1) were implemented as lists associated to v_i. Moreover, the door mappings $D2P(d_i)$, $D2P_{\sqsupset}(d_i)$, and $D2P_{\sqsubset}(d_i)$ were implemented as lists associated to the door d_i.

IDINDEX. The distance matrix and distance index matrix were implemented as 2D arrays.

CINDEX. Since the partitions in the datasets rarely intersect, we used an R-tree instead of R*-tree to index partitions while preserving roughly the same spatial search performance. We set the tree fan-out to 20 as suggested in a previous work [37]. Each partition's inter-partition links were maintained in an inner list.

IP-TREE and VIP-TREE. We set the minimum fanout to 2 for non-leaf tree nodes, as suggested in [31]. As each leaf node maintains the shortest distance for each pair of doors in it, the computation will be complicated if a leaf node contains too many C-Pars that each has many doors. Following work [31], we designate that each leaf node can only contain one crucial partition and regard a partition as crucial partition if its door number exceeds a threshold γ. Through tuning, we got optimal γ as 6, 4, 7, and 5 for SYN, MZB, HZM, and CPH, respectively.

5.4 Performance Evaluation Procedure

Concerning model construction and query processing, the following tasks are implemented to evaluate each model/index. For each task, a parameter is varied with others fixed to default. Table 6 lists all the evaluation settings. The code of following evaluation procedures and their query instances are also available online [1].

A **Model Construction.** For each model/index, we evaluate its (**a1**) model/index size and (**a2**) construction time. In this task, we vary the number of floors in synthetic datasets.

B **Query Processing.** We evaluate the search efficiency of a given query type. The metrics are (**b1**) running time, (**b2**) memory use, and (**b3**) number of visited doors (NVD) for SPDQ.

B1 **Effect of Floor Number n.** Using SYN with floor number n varied from 3 to 9, we test the search efficiency for each indoor spatial query algorithm.

B2 **Effect of Object Number $|O|$.** To test RQ and kNNQ, we vary $|O|$ from 500 to 2500 in all datasets.

B3 **Effect of Range Distance r.** We vary and test the augment r of RQ. In particular, we vary r from 200m to 1000m in SYN5, HZM and CPH, and from 20m to 100m in MZB.

B4 **Effect of k.** We vary and test kNNQ's augment k from 1 to 100 in all datasets.

B5 **Effect of Source-Target Distance s2t.** To test SPDQ, we vary s2t from 1100m to 1900m in SYN5, HZM, and CPH, and from 30m to 150m in MZB.

B6 **Effect of Topological Change.** We vary indoor topology by changing the door number from 840 to 1280 in SYN5 and obtain SYN5$^-$ and SYN5$^+$.

B7 **Effect of Hallway's Decomposition Method.** We use SYN5 and MZB with the derived datasets, SYN5^0, MZB^0 and MZB$^\Delta$.

6 RESULTS ANALYSIS

This section reports and analyzes the experimental results. All experiments are implemented in Java and run on a MAC with a 2.30GHz Intel i5 CPU and 16 GB memory.

6.1 Model/Index Construction

We vary the floor number n on SYN and obtain four variants SYN3, SYN5, SYN7, and SYN9. We construct the five model/indexes (cf. Section 3) and report their size and construction time in Figures 8 and 9. The cost of maintaining static objects is excluded as it is the same for all model/indexes.

- According to the results on SYN3 to SYN9 in Figure 8, each model/index's size increases steadily with a larger floor number.

When there are more doors and partitions, more storage space is needed to handle the indoor space.

- Among all, IDMODEL construction requires the least costs on storage (Figure 8) and time (Figure 9). This is because IDMODEL is extended based on a simple graph model and maintains only a small amount of geometric information locally. For large-scale and complex-topology spaces (e.g., SYN9, MZB, and HZM), IDMODEL has clearer advantages over the tree-based indexes (i.e., IP-TREE and VIP-TREE).
- As expected, IDINDEX always takes much time and storage to construct due to its global door-to-door distance precomputation. When there are many doors, it is difficult to fit the corresponding matrices in memory. In comparison, IP-TREE and VIP-TREE precompute less information and therefore their consumptions on time and storage are medium.
- In addition to maintaining the topology, CINDEX needs to construct a partition R-tree. Therefore, it incurs extra time and space overheads compared to IDMODEL.

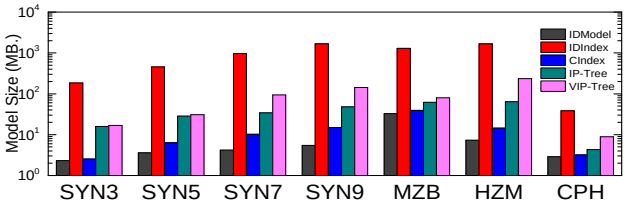


Figure 8: Model Size

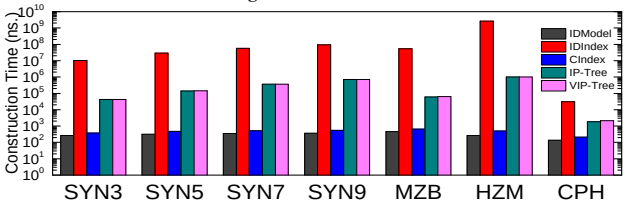


Figure 9: Construction Time

6.2 Query Processing

All results are averaged over 10 queries (cf. Section 5.2).

B1 Effect of Floor Number n (using SYN)

RQ and kNNQ: The query time and memory use for RQ are reported in Figures 10 and 11, respectively, and those for kNNQ are reported in Figures 12 and 13, respectively.

- For both query types, IDINDEX always runs fastest as shown in Figures 10 and 12, unaffected by the varying floor number n. The price behind this is to maintain the memory-resident distance matrices, which increases rapidly with n. Referring to Figures 11 and 13, when n grows to 9, IDINDEX requires up to 1600MB of memory on both queries.
- On each SYN dataset, IP-TREE and VIP-TREE need more time to complete the two queries. Through analysis, we found that the two indexes need to prune tree nodes when searching for qualified objects. In the absence of global door-to-door distances, they need a lot of on-the-fly calculations to get the shortest distance from a query point to a tree node. Being consistent with the complexity analysis in Table 4, VIP-TREE outperforms IP-TREE for both queries. However, due to the good scalability of the tree structure, both indexes' running time is relatively stable as shown in Figures 10 and 12.
- IDMODEL and CINDEX perform similarly, and their execution time increases with a larger n (Figures 10 and 12). When n increases, IDMODEL has a slight advantage as CINDEX costs more time in space pruning. In terms of memory overhead, the two indexes are almost the same.

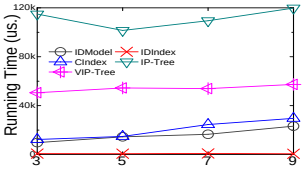


Figure 10: RQ Time vs. n

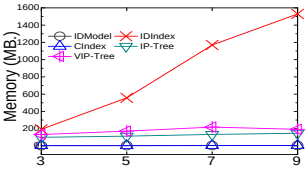


Figure 11: RQ Memory vs. n

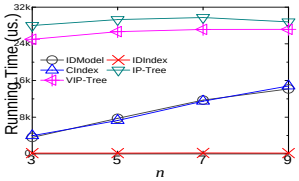


Figure 12: kNNQ Time vs. n

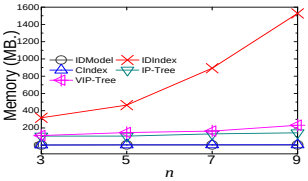


Figure 13: kNNQ Memory vs. n

SPDQ: The running time, memory use, and number of visited doors (NVD) are reported in Figures 14, 15, and 16, respectively.

- IDINDEX's running time and NVD are insensitive to the increasing floor number n. However, its memory use grows moderately as n increases. In the case of SPQ and SDQ, we recommend using IDINDEX when the door size is relatively small.
- In contrast to IDINDEX, the memory of IDMODEL and CINDEX is relatively stable (Figure 15), and their query performance deteriorates as the space scale increases (Figure 14).
- IP-TREE and VIP-TREE achieve clearly good performance on SPDQ, in both running time and memory use. Unlike IDINDEX that precomputes global door-to-door distances or IDMODEL and CINDEX that compute distances on the fly, IP-TREE and VIP-TREE cache relevant distance information only for those access doors on shortest paths. Thus, without degrading query performance, they only incur slightly more memory overhead than IDMODEL and CINDEX (Figures 14 and 15).

B2 Effect of Object Number $|O|$

RQ: With different sizes of O, the running time and memory use are reported in Figures 17 and 18, respectively.

- Algorithms based on different model/indexes are almost insensitive to $|O|$ in running time, implying that each can prune irrelevant objects effectively and stop searching early. A larger $|O|$ results in higher object density. This tends to increase the query processing time in general, as the query algorithms need to process larger object buckets. However, this impact is negligible according to the results in Figure 17. This implies that all model/indexes are good at pruning indoor partitions and thus object buckets when processing RQ.
- Referring to Figure 17, IDINDEX runs faster than others by several orders of magnitude in all datasets, thanks to its precomputed global door-to-door distances. However, it also requires memory an order of magnitude higher to store the distance matrix (Figure 18). A special case occurs on CPH (Figure 18(d)) that IP-TREE and VIP-TREE consume more memory than others. First, the door number of CPH is quite small such that the matrices of IDINDEX are not large. Second, as there are fewer access doors, IP-TREE/VIP-TREE involves heavy on-the-fly computations on distances between doors and non-leaf nodes and thus needs more memory for the intermediate results.
- On each dataset, IDMODEL and CINDEX incur almost the same execution time (see Figure 17), as they both use graph traversal to search for objects. Under complex indoor topology, CINDEX using R-tree does not have much advantage in spatial pruning.
- IP-TREE and VIP-TREE perform differently on different datasets. They outperform IDMODEL and CINDEX on MZB but are

worse on the others (see Figure 17). Recall that MZB features some C-Pars having up to 56 doors. In such a case, the efficiency of graph traversal is much lower than searching on the tree structure. On the contrary, when the number of candidate doors for the next hop is relatively small, the graph-based search algorithms are advantaged in range queries. Therefore, we recommend using IP-TREE/VIP-TREE to perform RQ in spaces with very large main corridors.

- Referring to Figure 17, VIP-TREE is generally faster than IP-TREE because of more cached distances. IP-TREE needs to compute more intermediate results on the fly. However, memory use is close between the two (see Figure 18).

*k*NNQ: Figures 19 and 20 report $|O|$'s impact on the time and memory costs, respectively. In general, each model/index's performance on *k*NNQ exhibits similar trend as that on RQ.

- Referring to Figure 19, the time cost of each algorithm on each dataset remains stable, showing that large object workloads (and high object density) have little effect on all models.
- On datasets with relatively large numbers of doors and partitions (i.e., SYN5, MZB, and HSM), IDINDEX runs faster by orders of magnitude. However, its memory use is clearly larger.
- On one-floor CPH with small numbers of doors and partitions, IP-TREE and VIP-TREE incur more running time as well as higher memory use (Figures 19(d) and 20(d)). However, they run faster on MZB (Figure 19(b)) in which many access doors exist due to many C-Pars (see Table 5).
- IDMODEL and CINDEX perform comparably as shown in Figures 19 and 20. Without a specially designed partition R-tree, IDMODEL achieves quite good object pruning due to the efficient distance mapping maintained in its edges and vertexes.

B3 Effect of Range Distance *r*

RQ: The time and memory costs with respect to varied *r* are reported in Figures 21 and 22, respectively.

- On SYN5, MZB, and HSM with complex indoor topology, IDINDEX's running time reported in Figure 21 increases slowly with a growing *r*. In contrast, on the simple-topology CPH, the advantage of IDINDEX over others is not marked.
- IDMODEL and CINDEX perform well on all datasets, except on MZB (Figure 21(b)) that has a large number of C-Pars. This again reflects the disadvantages of the graph-based traversal algorithms when dealing with this particular topology type. Nevertheless, through efficient node search and on-the-fly distance computation, these two model/indexes always have the smallest memory overhead.
- When increasing *r*, the running time of IP-TREE and VIP-TREE in Figure 21 increase steadily on all datasets. A larger *r* needs to consider a tree node farther from the node where the query point is located, and thus introduces more computations on the distance from a door to some non-leaf nodes. As the distance to the access door of each ancestor node is materialized at the leaf node, VIP-TREE runs faster than IP-TREE.

B4 Effect of *k*

*k*NNQ: The time and memory costs with respect to different *k* values are reported in Figures 23 and 24, respectively.

- Similar to increasing *r* value in RQ, increasing *k* leads to more search time by each model/index according to the results reported in Figure 23. Among them, IDINDEX's running time increases slowest. In addition, IP-TREE/VIP-TREE show exponential growth on SYN, HSM, and CPH. This is because the two indexes need to access the topologically far-away partitions and compute the distances to them on the fly when *k* is large.
- Considering both time and memory costs, IDMODEL and CINDEX achieve a good balance when searching for nearest neighbor objects (see Figures 23 and 24).

B5 Effect of Source-Target Distance s2t

SPDQ: The time cost, memory use, and NVD for different s2t values are reported in Figures 25, 26, and 27, respectively.

- IDINDEX runs the fastest and is not affected by s2t as reported in Figure 25. As only a small number of doors are required to process after the source point and before the target point, its NVD is always small (Figure 27). Nevertheless, its global distance matrix takes up a lot of memory (Figure 26).
- IDMODEL and CINDEX use the same graph search process. Note that because the Euclidean distance is no larger than the indoor distance, using R-tree to prune space by Euclidean distance does not really reduce the number of doors to visit. Therefore, the two models' NVDs in Figure 27 are almost the same. Also, as s2t increases, the candidate space becomes larger and the running time of the two becomes longer (see Figure 25).
- On MZB and HSM (Figure 25(b) and (c)), VIP-TREE achieves query performance comparable to IDINDEX that precomputes door-to-door distances. Both MZB and HSM are large-scale and have over 1000 doors. In the routing process based on VIP-TREE, the precomputed distances in non-leaf nodes greatly accelerate the expansion to the target point. Therefore, VIP-TREE is particularly suitable for the shortest path search in indoor spaces with complex structures.

Table 7: Results of RQ with Topological Change

Model	Time (us.)			Memory (MB.)		
	SYN5$^-$	SYN5	SYN5$^+$	SYN5$^-$	SYN5	SYN5$^+$
IDMODEL	11111	12770	19893	2	4	4
IDINDEX	308	417	910	289	520	809
CINDEX	13697	14877	19285	4	3	4
IP-TREE	29004	136600	574069	85	95	194
VIP-TREE	18008	58369	195583	75	171	220

Table 8: Results of *k*NNQ with Topological Change

Model	Time (us.)			Memory (MB.)		
	SYN5$^-$	SYN5	SYN5$^+$	SYN5$^-$	SYN5	SYN5$^+$
IDMODEL	5939	8180	10051	2	3	3
IDINDEX	165	146	181	469	573	1053
CINDEX	6865	8476	12998	4	4	4
IP-TREE	17341	36626	107798	74	89	139
VIP-TREE	14535	30145	75439	78	145	146

Table 9: Results of SPDQ with Topological Change

Model	Time (us.)			Memory (MB.)			NVD		
	SYN5$^-$	SYN5	SYN5$^+$	SYN5$^-$	SYN5	SYN5$^+$	SYN5$^-$	SYN5	SYN5$^+$
IDMODEL	23009	33213	35522	58	59	91	6946	10074	11426
IDINDEX	40	65	79	182	416	748	6	8	9
CINDEX	20219	31635	40408	51	63	99	6946	10074	11426
IP-TREE	3717	6398	7252	43	44	74	236	843	1455
VIP-TREE	2349	2369	2493	55	43	105	52	61	90

B6 Effect of Topological Change

RQ and *k*NNQ: The time cost and memory use with respect to topology characteristics are reported in Tables 7 and 8 respectively.

- IDINDEX runs fastest, but it needs large memory to store the door-to-door distance matrix. With increasing number of doors, its time cost and memory use increase steadily.
- IDMODEL and CINDEX use the smallest memory when processing RQ and *k*NNQ. Regarding the time cost, they perform medium. When the topology becomes more complex, the memory use keeps stable and the time cost increases slightly.
- IP-TREE and VIP-TREE cost more time to process RQ and *k*NNQ. Moreover, when the topology becomes more complex, the time cost rises rapidly. E.g., RQ's time cost using IP-TREE grows nearly 20 times from SYN5$^-$ to SYN5$^+$.

SPDQ: The time cost, memory use and NVD with respect to different topology characteristics are reported in Table 9.

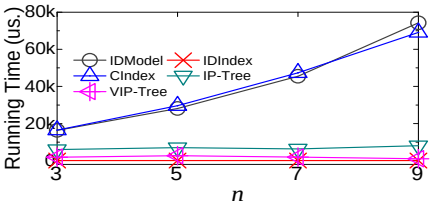


Figure 14: SPDQ Time vs. n

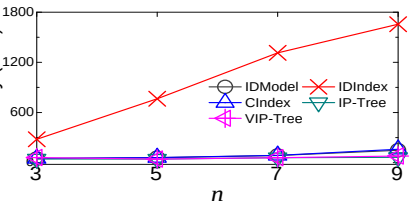


Figure 15: SPDQ Memory vs. n

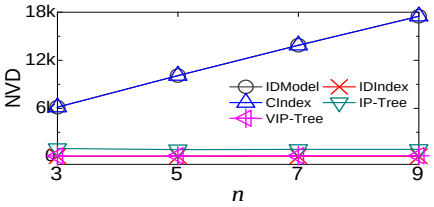


Figure 16: NVD in SPDQ vs. n

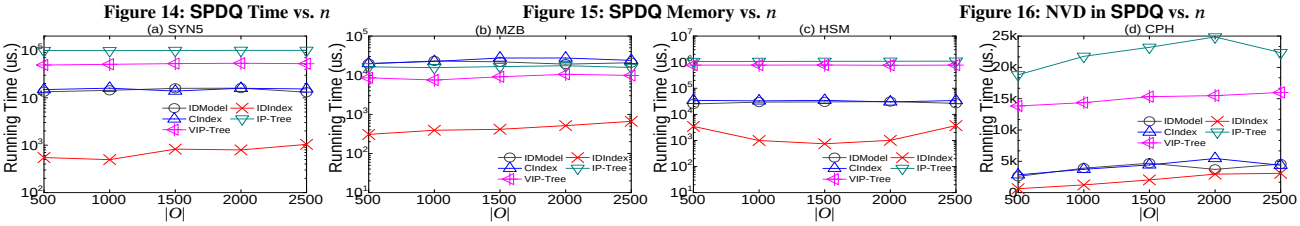


Figure 17: RQ Time vs. $|O|$

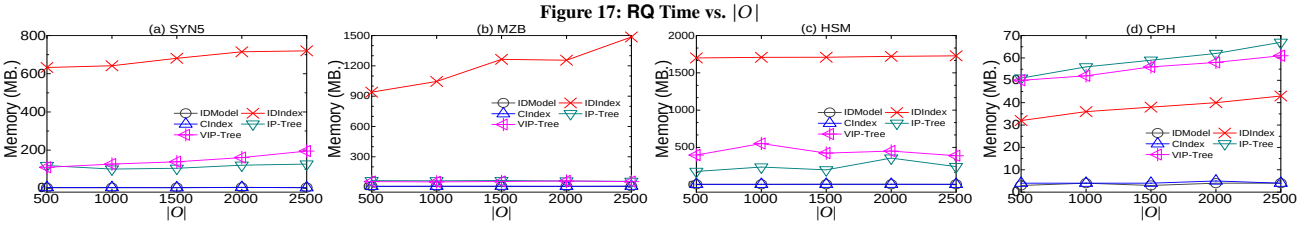


Figure 18: RQ Memory vs. $|O|$

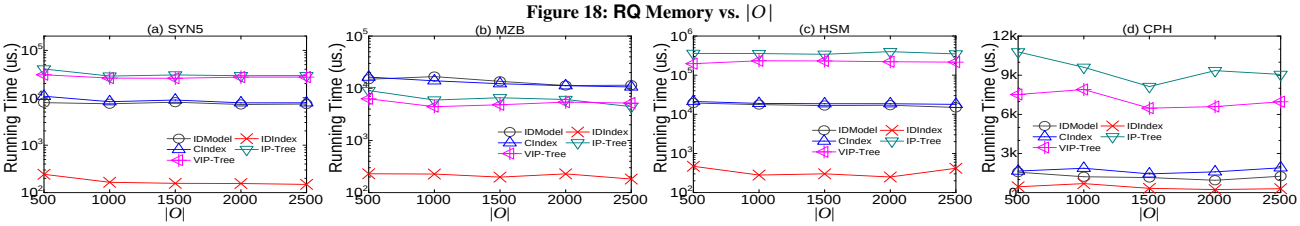


Figure 19: kNNQ Time vs. $|O|$

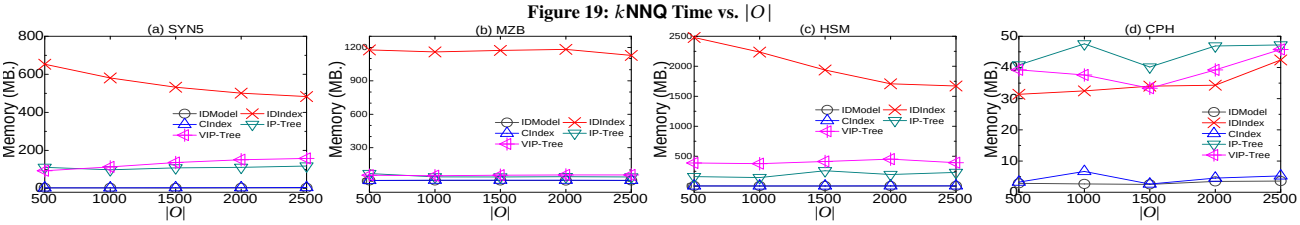


Figure 20: kNNQ Memory vs. $|O|$

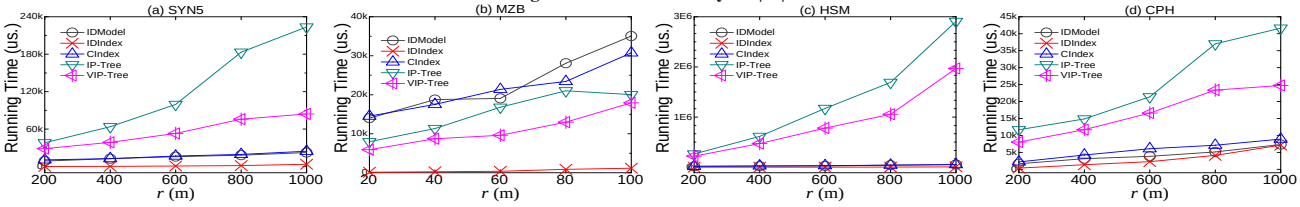


Figure 21: RQ Time vs. r

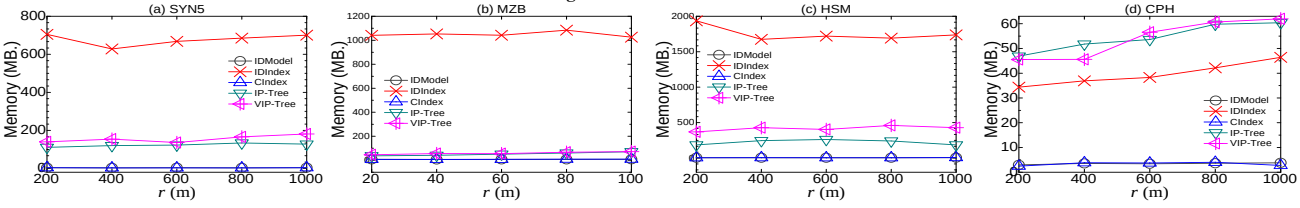


Figure 22: RQ Memory vs. r

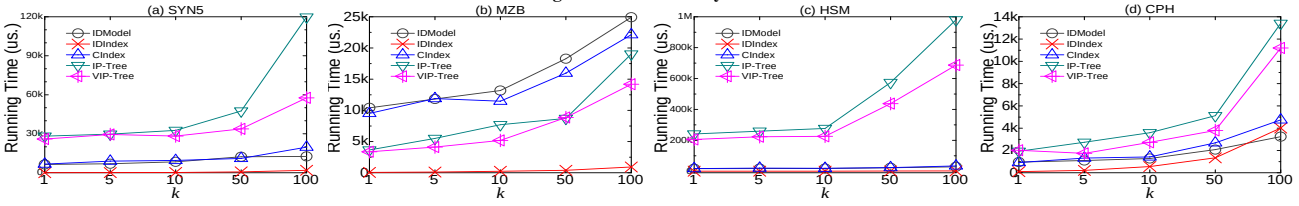


Figure 23: kNNQ Time vs. k

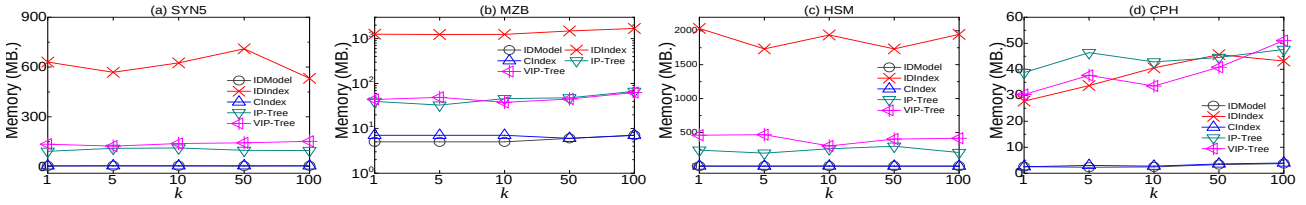


Figure 24: kNNQ Memory vs. k

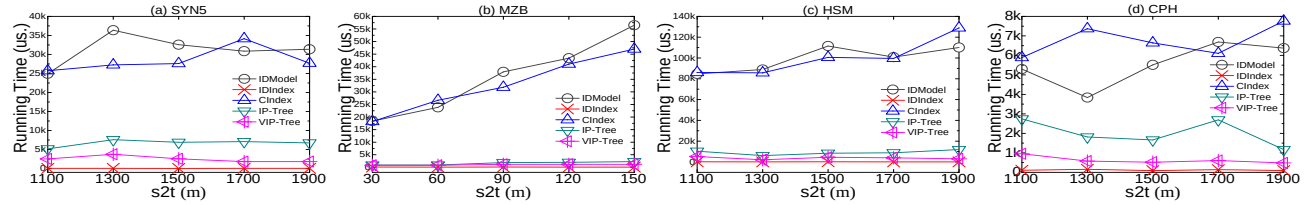


Figure 25: SPDQ Time vs. s2t

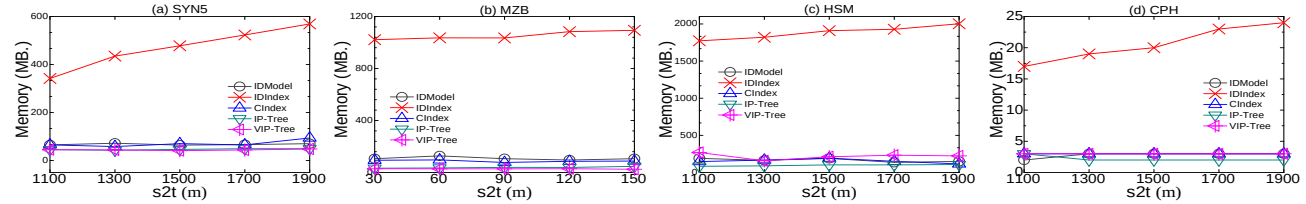


Figure 26: SPDQ Memory vs. s2t

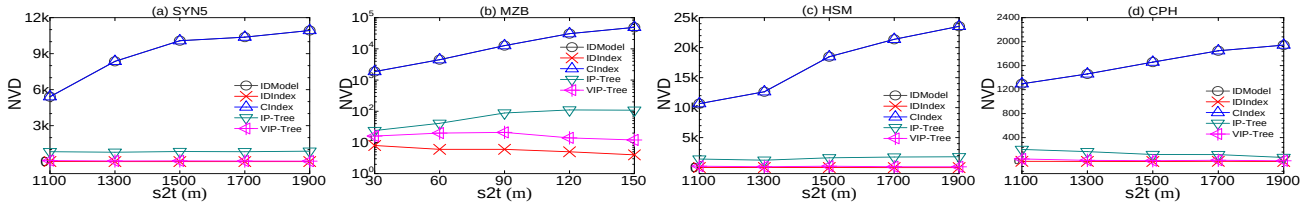


Figure 27: NVD in SPDQ vs. s2t

- Like in the other cases, IDINDEX performs best in terms of the time cost but costs most memory compared with others. When the topology becomes complex, IDINDEX's time cost increases relatively slightly, while the memory use grows fast.
- IP-TREE/VIP-TREE perform best with relatively less time cost and smaller memory use. For time cost, VIP-TREE always outperforms IP-TREE because of the extra precomputation, but it needs more memory. With the doors increasing, their time and memory costs rise slightly.
- IDMODEL and CINDEX performs worst in both time and memory costs because they have to visit many doors in search.

Table 10: Results of RQ with Decomposition Method

Model	Time (us.)					Memory (MB.)				
	SYN5^0	SYN5	MZB^0	MZB	MZB$^\Delta$	SYN5^0	SYN5	MZB^0	MZB	MZB$^\Delta$
IDMODEL	9695	14999	24065	23527	18917	13	3	12	7	5
IDINDEX	460	704	471	439	349	414	437	815	841	1855
CINDEX	11283	15859	21840	21351	20267	12	4	11	8	5
IP-TREE	8923	123076	7957	17215	26110	88	92	61	58	76
VIP-TREE	6808	57988	4476	11079	19181	111	150	62	59	78

Table 11: Results of kNNQ with Decomposition Method

Model	Time (us.)					Memory (MB.)				
	SYN5^0	SYN5	MZB^0	MZB	MZB$^\Delta$	SYN5^0	SYN5	MZB^0	MZB	MZB$^\Delta$
IDMODEL	4773	9240	14318	14224	12828	9	3	12	6	4
IDINDEX	143	160	180	185	197	461	457	679	796	974
CINDEX	4907	9294	13115	13328	13225	16	4	11	8	6
IP-TREE	7272	33693	3904	7315	10369	112	114	36	36	52
VIP-TREE	6877	24522	3556	5207	7502	117	139	43	55	59

B7 Effect of Decomposition Methods for Hallways

RQ, kNNQ and SPDQ: For RQ and kNNQ, their time cost and memory use with respect to different decomposition methods are reported in Tables 10 and 11. For SPDQ, its time cost, memory use and NVD are reported in Table 12.

- IDINDEX runs fastest when processing RQ and kNNQ but uses most memory. When hallways are decomposed into more partitions, IDINDEX's time cost keeps nearly stable but its memory cost increases. This is because there are more doors connecting increased numbers of partitions, which leads to more door-to-door pairs stored in the distance matrix.
- IDMODEL and CINDEX use the least memory but runs slowest. With more partitions, both time cost and memory use decrease because hallways are decomposed into more partitions each having less doors to process.
- IP-TREE and VIP-TREE perform best considering both time cost and memory use. However, when hallways are decomposed into more partitions, the two methods need more time and memory to process RQ and kNNQ. Regarding the performance in RQ, IP-TREE and VIP-TREE cost more time than IDMODEL. There are more nodes in IP-TREE and VIP-TREE when hallways are decomposed into more partitions, which entails more on-the-fly computations to prune tree nodes when processing RQ and kNNQ. Moreover, the time cost of IP-TREE and VIP-TREE rises faster when processing RQ and kNNQ than processing SPDQ. That is because there is some extra cost to prune nodes when processing RQ and kNNQ. As the nodes increase, this extra cost increases fast.

6.3 Summary of Findings

We summarize all five model/indexes' performance in Table 13 where more stars imply a better performance (lower cost). IDMODEL incurs minimum time and space costs in construction. It

Table 12: Results of SPDQ with Decomposition Method

Model	Time (us.)					Memory (MB.)					NVD				
	$SYN5^0$	SYN5	MZB^0	MZB	MZB^Δ	$SYN5^0$	SYN5	MZB^0	MZB	MZB^Δ	$SYN5^0$	SYN5	MZB^0	MZB	MZB^Δ
IDMODEL	31242	31855	36220	33503	32396	44	63	54	73	92	82574	10074	24877	12718	4243
IDINDEX	138	75	71	69	73	388	396	1096	1273	1489	58	8	22	9	8
CINDEX	32823	26900	35307	33238	31806	43	65	52	97	110	82574	10074	24877	12718	4243
IP-TREE	1610	7523	1252	1893	3257	59	44	31	39	44	416	843	97	87	139
VIP-TREE	856	2379	1091	1126	1474	64	42	54	39	55	136	61	37	24	26

works well for RQ and kNNQ, and its performance for SPQ/SDQ even improves when hallways are decomposed into more partitions. IDINDEX runs fastest for all types of indoor spatial queries while requiring significantly large time to construct offline and high memory consumptions during search. CINDEX performs only comparably to IDMODEL when processing the queries. IP-TREE and VIP-TREE are optimized for SPQ/SDQ tasks—they stand out when there are many C-Pars connected by so-called access doors; they decline when decomposition reduces C-Pars.

In short, IDINDEX is preferred for small-scale spaces. VIP-TREE is recommended if routing is the task or the space accommodates many C-Pars. Otherwise, IDMODEL is recommended for non-routing queries due to its low construction cost and good balance between storage and query time costs.

Table 13: Summary of Findings

Model	Construction Cost		RQ/kNNQ Search		SPQ/SDQ Search	
	Model Size	Time	Memory	Time	Memory	Time
IDMODEL	★★★★★	★★★★★	★★★★★	★★★	★★★★★	★
IDINDEX	★	★	★	★★★★★	★	★★★★★
CINDEX	★★★★	★★★★	★★★★★	★★★	★★★★★	★
IP-TREE	★★★	★★★	★★★★	★	★★★★	★★★
VIP-TREE	★★	★★★	★★★	★★	★★★	★★★★

7 CONCLUSION AND FUTURE WORK

This work reports on an extensive experimental evaluation of five indoor space model/indexes that support four typical indoor spatial queries. Our evaluation concerns the costs in model/index construction and query processing using a model/index. By analyzing the results, we summarize the pros and cons of all techniques and suggest the best choice for typical scenarios.

For future work, changes to existing methods may improve their performance. First, heuristics like A* and IDA* algorithms can replace the Dijkstra-based expansion in IDMODEL and CINDEX to speed up SPDQ processing. Second, intra-partition indexes like grids can be combined with CINDEX and IP-TREE/VIP-TREE to achieve local object pruning in processing RQ and kNNQ. Third, strategies to select crucial doors/partitions can be developed to reduce the storage of door-to-door distances in CINDEX and IP-TREE/VIP-TREE while preserving their search efficiency.

Acknowledgement. This work was supported by IRFD (No. 8022-00366B), ARC (No. FT180100140 and DP180103411), the Key R&D Program (Zhejiang, China) (No. 2021C009) and NSFC (No. 62050099).

REFERENCES

[1] https://github.com/indoorLBS/ISQEA.
[2] Tanvir Ahmed, Torben Bach Pedersen, and Hua Lu. 2014. Finding dense locations in indoor tracking data. In *MDM*. 189–194.
[3] Tanvir Ahmed, Torben Bach Pedersen, and Hua Lu. 2017. Finding dense locations in symbolic indoor tracking data: modeling, indexing, and processing. *GeoInformatica* 21, 1, 119–150.
[4] S. Alamri, D. Taniar, and M. Safar. 2012. Indexing Moving Objects in Indoor Cellular Space. In *NBiS*. 38–44.
[5] Douglas G Altman and J Martin Bland. 1994. Statistics notes: quartiles, quintiles, centiles, and other quantiles. *BMJ* 309, 6960, 996–996.
[6] Anahid Basiri, Elena Simona Lohan, Terry Moore, and et al. 2017. Indoor location based services challenges, requirements and usability of current solutions. *Computer Science Review* 24, 1–12.
[7] Thomas Becker, Claus Nagel, and Thomas H Kolbe. 2009. A multilayered space-event model for navigation in indoor spaces. In *3D geo-information sciences*. 61–77.
[8] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. 1990. The R*-tree: an efficient and robust access method for points and rectangles. 322–331.
[9] Muhammad Aamir Cheema. 2018. Indoor location-based services: challenges and opportunities. *SIGSPATIAL Special* 10, 2, 10–17.
[10] Lisi Chen, Gao Cong, Christian S Jensen, and Dingming Wu. 2013. Spatial keyword query processing: an experimental evaluation. *PVLDB* 6, 3, 217–228.
[11] Constantinos Costa, Xiaoyu Ge, and Panos Chrysanthis. 2019. CAPRIO: Context-Aware Path Recommendation Exploiting Indoor and Outdoor Information. In *MDM*. 431–436.
[12] Matthias Delafontaine, Mathias Versichele, Tijs Neutens, and Nico Van de Weghe. 2012. Analysing spatiotemporal sequences in Bluetooth tracking data. *Applied Geography* 34, 659–668.
[13] Zijin Feng, Tiantian Liu, Huan Li, Hua Lu, Lidan Shou, and Jianliang Xu. 2020. Indoor Top-k Keyword-aware Routing Query. In *ICDE*. 1213–1224.
[14] Marcus Goetz and Alexander Zipf. 2011. Formal definition of a user-adaptive and length-optimal routing graph for complex indoor environments. *Geo-Spatial Information Science* 14, 2, 119–128.
[15] Christian S Jensen, Hua Lu, and Bin Yang. 2009. Indexing the trajectories of moving objects in symbolic indoor space. In *SSTD*. 208–227.
[16] Xinlong Jiang, Yunbing Xing, Yiqiang Chen, Yang Gu, and Junfa Liu. 2019. Indoor Trajectory Restoration Method Based on PoI Relation Constraints. In *SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI*. 439–446.
[17] Peiquan Jin, Tong Cui, Qian Wang, and Christian S Jensen. 2016. Effective similarity search on indoor moving-object trajectories. In *DASFAA*. 181–197.
[18] Donald B Johnson. 1973. A note on Dijkstra's shortest path algorithm. *J. ACM* 20, 3 (1973), 385–388.
[19] YongHee Kim, HaRim Jung, JaeHee Jang, and Ung-Mo Kim. 2016. An efficient grid index for moving objects in indoor environments. In *IMCOM*. 1–4.
[20] Jiyeong Lee. 2004. A spatial access-oriented implementation of a 3-D GIS topological data model for urban entities. *GeoInformatica* 8, 3, 237–264.
[21] Huan Li, Hua Lu, Muhammad Aamir Cheema, Lidan Shou, and Gang Chen. 2020. Indoor mobility semantics annotation using coupled conditional Markov networks. In *ICDE*. 1441–1452.
[22] Huan Li, Hua Lu, Lidan Shou, Gang Chen, and Ke Chen. 2018. In search of indoor dense regions: An approach using indoor positioning data. *IEEE Trans. Knowl. Data Eng.* 30, 8, 1481–1495.
[23] Huan Li, Hua Lu, Lidan Shou, Gang Chen, and Ke Chen. 2019. Finding most popular indoor semantic locations using uncertain mobility data. *IEEE Trans. Knowl. Data Eng.* 31, 11, 2108–2123.
[24] Hui Lin, Ling Peng, Si Chen, Tianyue Liu, and Tianhe Chi. 2016. Indexing for moving objects in multi-floor indoor spaces that supports complex semantic queries. *ISPRS International Journal of Geo-Information* 5, 10, 176.
[25] Tiantian Liu, Zijin Feng, Huan Li, Hua Lu, Muhammad Aamir Cheema, Hong Cheng, and Jianliang Xu. 2020. Shortest Path Queries for Indoor Venues with Temporal Variations. In *ICDE*. 2014–2017.
[26] Tiantian Liu, Huan Li, Hua Lu, Muhammad Aamir Cheema, and Lidan Shou. 2020. An Experimental Analysis of Indoor Spatial Queries: Modeling, Indexing, and Processing. arXiv:2010.03910
[27] Hua Lu, Xin Cao, and Christian S Jensen. 2012. A foundation for efficient indoor distance-aware query processing. In *ICDE*. 438–449.
[28] Hua Lu, Chenjuan Guo, Bin Yang, and Christian S Jensen. 2016. Finding Frequently Visited Indoor POIs Using Symbolic Indoor Tracking Data. In *EDBT*. 449–460.
[29] Hua Lu, Bin Yang, and Christian S Jensen. 2011. Spatio-temporal joins on symbolic indoor tracking data. In *ICDE*. 816–827.
[30] Chaluka Salgado. 2018. Keyword-aware skyline routes search in indoor venues. In *SIGSPATIAL-ISA*. 25–31.
[31] Zhou Shao, Muhammad Aamir Cheema, David Taniar, and Hua Lu. 2016. Vip-tree: an effective index for indoor spatial queries. *PVLDB* 10, 4, 325–336.
[32] Zhou Shao, Muhammad Aamir Cheema, David Taniar, Hua Lu, and Shiyu Yang. 2020. Efficiently Processing Spatial and Keyword Queries in Indoor Venues. *IEEE Trans. Knowl. Data Eng.*.
[33] Nico Sun, Erfu Yang, Jonathan Corney, and Yi Chen. 2019. Semantic path planning for indoor navigation and household tasks. In *TAROS*. 191–201.
[34] Zhifeng Wang, Heng Xie, Zeqin Lin, Tao Wen, Chenglong Guo, and Haichu Chen. 2020. The Robot Path Planning Algorithm In Indoor Environment. In *IECON*. 5350–5355.
[35] Emily Whiting, Jonathan Battat, and Seth Teller. 2007. Topology of urban environments. In *CAADFutures*. 114–128.
[36] Michael Worboys. 2011. Modeling indoor space. In *SIGSPATIAL Workshop on Indoor Spatial Awareness*. 1–6.
[37] Xike Xie, Hua Lu, and Torben Bach Pedersen. 2013. Efficient distance-aware query evaluation on indoor moving objects. In *ICDE*. 434–445.
[38] Xike Xie, Hua Lu, and Torben Bach Pedersen. 2014. Distance-aware join for indoor moving objects. *IEEE Trans. Knowl. Data Eng.* 27, 2, 428–442.
[39] Bin Yang, Hua Lu, and Christian S Jensen. 2009. Scalable continuous range monitoring of moving objects in symbolic indoor space. In *CIKM*. 671–680.
[40] Bin Yang, Hua Lu, and Christian S Jensen. 2010. Probabilistic threshold k nearest neighbor queries over moving objects in symbolic indoor space. In *EDBT*. 335–346.
[41] Jiao Yu, Wei-Shinn Ku, Min-Te Sun, and Hua Lu. 2013. An RFID and particle filter-based indoor spatial query evaluation system. In *EDBT*. 263–274.