Spatial Queries for Indoor Location-based Services

Liu, Tiantian

DOI (link to publication from Publisher):
10.54337/aau470863740

Publication date:
2022

Document Version
Publisher's PDF, also known as Version of record

SPATIAL QUERIES FOR INDOOR LOCATION-BASED SERVICES

BY
TIANTIAN LIU

DISSERTATION SUBMITTED 2022

AALBORG UNIVERSITY
DENMARK

Spatial Queries for Indoor Location-based Services

Ph.D. Dissertation

Tiantian Liu

Dissertation submitted February, 2022

Dissertation submitted: February, 2022

PhD supervisor: Professor Hua Lu
Aalborg University, Denmark

PhD committee: Associate Professor Simonas Saltenis (chairman)
Aalborg University, Denmark

Professor Christophe Claramunt
Naval Academy Research Institute, France

Professor Lars Kulik
University of Melbourne, Australia

PhD Series: Technical Faculty of IT and Design, Aalborg University

Department: Department of Computer Science

ISSN (online): 2446-1628
ISBN (online): 978-87-7573-938-7

Published by:
Aalborg University Press
Kroghstræde 3
DK – 9220 Aalborg Ø
Phone: +45 99407140
aauf@forlag.aau.dk
forlag.aau.dk

Printed in Denmark by Rosendahls, 2022

Abstract

People have many activities and spend a lot of time in indoor spaces such as airports, shopping malls, and office buildings. There is an increasing demand for indoor spatial queries which support indoor location-based services. To support the general indoor spatial queries, different techniques have been proposed. However, a comprehensive experimental study on these existing techniques is still missing and we have to further study different indoor spatial queries to facilitate people in different scenarios in reality.

In this thesis, we focus on the following specific works. 1) Experimental study on existing proposals for indoor spatial queries. 2) Indoor Keyword-aware Routing Query. 3) Indoor Temporal-variation aware Routing Query. 4) Indoor Crowd-aware Routing Query.

First, we survey five existing models/indexes for indoor spatial queries. We do a comprehensive comparison for their structure, corresponding algorithm characteristics, and their space and complexities. To evaluate the performance of these techniques, we design an in-depth benchmark including tasks and performance metrics. We conduct extensive experiments on both real and synthetic datasets. From the results, we analyze the pros and cons of different techniques.

Second, we study an indoor keyword-aware routing query that considers the textual information. Given two indoor points s and t, an Indoor top-k Keyword-aware Routing Query (IKRQ) returns k s-to-t routes that do not exceed a given distance constraint but have optimal ranking scores integrating keyword relevance and spatial distance. We design a fundamental model to organize textual information and spatial information. We propose a method to calculate the ranking score which integrates keyword relevance and indoor spatial distance. We design two algorithms with several pruning rules to process IKRQ. We conduct extensive experiments on synthetic and real data. The results show our proposals are efficient.

Third, we study an indoor temporal-variation aware routing query that takes into account temporal variations, e.g., the open and close times associated with entities like doors and rooms. Given two indoor static points p_s and p_t, and a current timestamp t, an Indoor Temporal-variation aware Shortest Path Query (ITSPQ) returns the valid shortest path from p_s to p_t based on the up-to-date indoor topology at the query time t. To answer an ITSPQ, we design the Indoor Temporal-variation Graph (IT-

Graph) to organize indoor spatial information with temporal variations. Two corresponding algorithms are proposed to process the query. The main difference between the two algorithms is that one checks a door's accessibility synchronously while the other one check that asynchronously. To speed up query processing, we propose the Indoor Temporal-variation Index (IT-Index) which maintains dynamic door-to-door distances. To evaluate the proposed techniques, we conduct extensive experiments. The result demonstrates our IT-Index based method is the most efficient.

Finally, we study two indoor crowd-aware routing queries that consider the influence of indoor crowd when finding a route. Give two indoor points p_s and p_t, and a timestamp t, an Indoor Crowd-Aware Fastest Path Query (`FPQ`) returns the fastest path considering the influence of crowds, whereas an Indoor Least Crowded Path Query (`LCPQ`) returns a path encountering the least objects en route. We propose a unified framework that consists of three main components including an indoor crowd model, a time-evolving population estimator, and two exact and two approximate algorithms to process each type of query. Extensive experiments are conducted, and the results show the efficiency and scalability of the proposed framework and algorithms.

Resumé

Folk bruger meget tid i indendørs rum såsom lufthavne, indkøbscentre og kontorbygninger. Der er en stigende efterspørgsel efter indendørs rumlige forespørgsler, som understøtter indendørs lokations baserede services. For at understøtte de generelle indendørs rumlige forespørgsler er forskellige teknikker blevet foreslået. En omfattende eksperimentel undersøgelse af disse eksisterende teknikker mangler dog stadig, og vi er nødt til yderligere at forske forskellige indendørs rumlige forespørgsler for at lette mennesker i forskellige scenarier i virkeligheden.

Afhandlingen fokuserer på følgende specifikke emner. 1) Eksperimentel undersøgelse af eksisterende forslag til indendørs rumlige forespørgsler. 2) Indendørs søgeordsbevidst ruteforespørgsel. 3) Indendørs tidsvariationbevidst routing forespørgsel. 4) Indendørs crowd-aware routing-forespørgsel.

Først undersøger vi fem eksisterende modeller/indekser for indendørs rumlige forespørgsler. Vi laver en omfattende sammenligning for deres struktur, tilsvarende algoritmekarakteristika og deres plads ogtid kompleksitet. For at evaluere ydeevnen af teknikker designer vi en dybdegående benchmark, herunder opgaver og præstationsmålinger. Vi udfører omfattende eksperimenter på både rigtige og syntetiske datasæt. Ud fra resultaterne analyserer vi fordele og ulemper ved forskellige teknikker.

For det andet forske vi en indendørs søgeordsbevidst routingforespørgsel, der tager hensyn til tekstinformationen. For to indendørs punkter s og t returnerer en Indoor top-k Keyword-aware Routing Query (IKRQ) k s-to-t ruter, der ikke overskrider en given afstandsbegrænsning, men har optimale rangeringsscore, der integrerer søgeordsrelevans og rumlig afstand. Vi designer en grundlæggende model til at organisere tekstinformation og rumlig information. Vi foreslår en metode til at beregne rangeringsscore, som integrerer søgeordsrelevans og indendørs rumlig afstand. Vi designer to algoritmer med flere beskæringsregler til at behandle IKRQ. Vi udfører omfattende eksperimenter med syntetiske og rigtige data. Resultaterne viser, at vores forslag er effektive.

For det tredje forske vi en indendørs tidsvariationsbevidst routing-forespørgsel, der tager højde for tidsmæssige variationer, f.eks. de åbne og lukketider, der er forbundet med enheder som døre og værelser. For to indendørs statiske punkter p_s og p_t og et aktuelt tidsstempel t returnerer en Indoor Temporal-variation aware Shortest Path Query (ITSPQ) den gyldige korteste rute fra p_s til p_t baseret på up-to-date in-

dendørstopologi på forespørgselstidspunktet t. For at besvare en ITSPQ designer vi Indoor Temporal-variation Graph (IT-Graph) for at organisere indendørs rumlig information med tidsmæssige variationer. To tilsvarende algoritmer foreslås til at behandle forespørgslen. Den største forskel mellem de to algoritmer er, at den ene kontrollerer en døres tilgængelighed synkront, mens den anden tjekker den asynkront. For at fremskynde forespørgselsbehandlingen foreslår vi Indendørs Temporal-variationsindeks (IT-indeks), som opretholder dynamiske dør-til-dørafstande. For at evaluere de foreslåede teknikker udfører vi omfattende eksperimenter. Resultatet viser, at vores IT-indeksbaserede metode er den mest effektive.

Til sidst forske vi to indendørs crowd-aware routing-forespørgsler, der overvejer indflydelsen fra indendørs crowd, når de finder en rute. For to indendørs punkter p_s og p_t, og et tidsstempel t returnerer en Indoor Crowd-Aware Fastest Path Query (`FPQ`) den hurtigste rute i betragtning af påvirkningen fra menneskemængder, mens en Indoor Least Crowded Path Query (`LCPQ`) returnerer en rute, der støder på de mindste objekter undervejs. Vi foreslår en samlet ramme, der består af tre hovedkomponenter, herunder en indendørs publikumsmodel, en tidsudviklende befolkningsvurdering og to nøjagtige og to omtrentlige algoritmer til at behandle hver type forespørgsel. Der udføres omfattende eksperimenter, og resultaterne viser effektiviteten og skalerbarheden af de foreslåede rammer og algoritmer.

Contents

Acknowledgement

First, I would like to express my gratitude to my Ph.D. supervisor, Professor Hua Lu, for giving me the opportunity to work with him. He is an excellent researcher and a nice supervisor who always gives me professional guidance, warm encouragement, and enormous support. He is always there to help and support me whenever I need him. I am very grateful for his constructive suggestions and inspiring ideas. Under his patient supervision, I have improved a lot in the research area. It is my great honor to be one of his Ph.D. students. If it is possible I would like to continue to work with him in my future career.

Second, I would like to thank Assistant Professor Huan Li, who helped me a lot during my Ph.D. study. Without his professional and detailed advice, I cannot imagine I can have these publications and thesis. We had a lot of discussions regarding my research work and I was always inspired during the discussion. I am impressed by his rigorous approach and strong logical thinking. I learned a lot from him during our cooperation.

Third, I would like to thank Associate Professor Muhammad Aamir Cheema who hosted me as a visiting student at Monash University. Due to the Covid-19, I could not study at Monash University physically, but it was a special experience for me. I appreciate his support and a great help to my research work.

Fourth, I would like to thank all my co-authors, Zijin Feng, Doctor Harry Kai-Ho Chan, Professor Lidan Shou, Professor Jianliang Xu, and Professor Hong Cheng, for giving me the chance to work with them and learn from them. I want to thank all my colleagues at Daisy who provide a friendly working environment. I also want to thank all administrative staff at AAU for their valuable support regarding administrative issues during my Ph.D. study.

Fifth, I would like to thank the Independent Research Fund Denmark and the Department of Computer Science, Aalborg University for funding me to finish my Ph.D. study.

Finally, I would like to thank my family and friends, especially my parents and my husband who always encourage me with warm words. Thank them for their love and patience. Thank all my friends who let me feel I am not lonely.

Tiantian Liu
Aalborg University, February 26, 2022

Thesis Details

Thesis Title: Spatial Queries for Indoor Location-based Services
Ph.D. Student: Tiantian Liu
Supervisor: Prof. Hua Lu, Aalborg University

The main body of the thesis consists of the following papers.

A. **Tiantian Liu**, Huan Li, Hua Lu, Muhammad Aamir Cheema, and Lidan Shou, "Indoor spatial queries: Modeling, indexing, and processing," in 24th International Conference on Extending Database Technology (EDBT), 2021, pp. 181–192.

B. Zijin Feng, **Tiantian Liu**, Huan Li, Hua Lu, Lidan Shou, and Jianliang Xu, "Indoor top-k keyword-aware routing query," in 2020 IEEE 36th International Conference on Data Engineering (ICDE). IEEE, 2020, pp. 1213–1224.

C. **Tiantian Liu**, Zijin Feng, Huan Li, Hua Lu, Muhammad Aamir Cheema, Hong Cheng, and Jianliang Xu, "Towards indoor temporal-variation aware shortest path query," IEEE Transactions on Knowledge and Data Engineering (TKDE), 2021.

D. **Tiantian Liu**, Huan Li, Hua Lu, Muhammad Aamir Cheema, and Lidan Shou, "Towards crowd-aware indoor path planning," Proc. VLDB Endow. (PVLDB), vol. 14, no. 8, pp. 1365–1377, 2021.

In addition to the above papers, I have co-authored the following five papers as part of my Ph.D. studies, which are not included in the thesis.

E. **Tiantian Liu**, Zijin Feng, Huan Li, Hua Lu, Muhammad Aamir Cheema, Hong Cheng, and Jianliang Xu, "Shortest path queries for indoor venues with temporal variations," in 2020 IEEE 36th International Conference on Data Engineering (ICDE). IEEE, 2020, pp. 2014–2017 (Poster Paper).

F. Harry Kai-Ho Chan, **Tiantian Liu**, Huan Li, and Hua Lu. "Time-Constrained Indoor Keyword-aware Routing," Proceedings of the 17th International Symposium on Spatial and Temporal Databases (SSTD), 2021: 74-84.

G. **Tiantian Liu**, Zijin Feng, Huan Li, Hua Lu, Lidan Shou, and Jianliang Xu, "IKAROS: An Indoor Keyword-Aware Routing System," accepted by 2022 IEEE 38th International Conference on Data Engineering (ICDE) (Demo Paper).

H. **Tiantian Liu**, Huan Li, Hua Lu, and Muhammad Aamir Cheema. Contact "Contact Tracing over Uncertain Indoor Positioning Data," under review.

I. Harry Kai-Ho Chan, **Tiantian Liu**, Huan Li, and Hua Lu. "Time-Constrained Indoor Keyword-aware Routing: Foundations and Extensions," under review.

This thesis has been submitted for assessment in partial fulfillment of the Ph.D. degree. The thesis is based on the submitted or published scientific papers listed above. Parts of the content of the papers in the main body of the thesis are used directly or indirectly in the extended summary part of the thesis. As part of the assessment, co-author statements have been made available to the assessment committee and are also available at the Faculty. The permission for using the published and accepted articles in the thesis have been obtained from the corresponding publishers with the condition that they are cited and copyright are placed prominently in the references. In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of Aalborg University's products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to `http://www.ieee.org/publications_standards/publications/rights/rights_link.html` to learn how to obtain a License from RightsLink.

Part I

Thesis Summary

Chapter 1

Introduction

1.1 Background and Motivation

With the acceleration of urbanization, indoor space has become increasingly large and complex, resulting in a lot of indoor LBSs needs. For instance, passengers may want to query the fastest path to their boarding gate in an airport. Besides, people may want to know the nearest cafe with a high rating in a shopping mall. In a hospital, patients want to find a path to the doctor's office. Indoor spatial queries are the basics to support relevant applications in different indoor scenarios.

Some works study the indoor model, index, and query [19, 24, 25]. Nonetheless, there is still no overall experimental study on these existing techniques. Moreover, existing works in indoor spatial query [19, 24, 25] cannot facilitate people in all scenarios in reality. More aspects should be considered, and these are shown in Fig. 1.1.

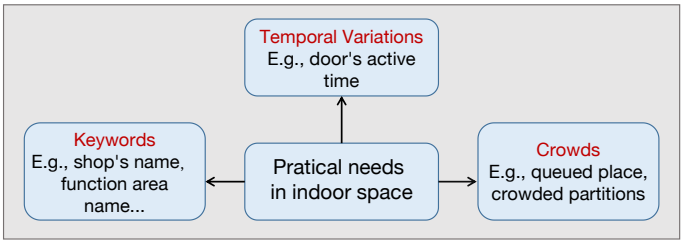


Fig. 1.1: Pratical Needs in Indoor Space

- **Keywords.** In real life, people have the demand of keyword-aware routing queries, in particular, sometimes people have to across a strange large indoor building. For example, Jesper wants to take a flight in Copenhagen Airport. Before he reaches the boarding gate, he wants to draw some cash at an ATM, have lunch, and buy some Danish cookies as a gift for friends. In this case,

it needs an appropriate route that can cover some keywords, such as "cash", "restaurant", and "cookies".

- **Temporal variations.** There are some temporal variations and restrictions in indoor venues in real life. For example, in a shopping mall, some doors may close in the evening which causes different indoor topology. Some indoor partitions like private offices in an office building will also result in different indoor topologies because we usually cannot pass through these kinds of partitions in reality.

- **Crowds.** Moving objects in indoor space may also influence how people choose an appropriate route. For instance, people's moving speed will become slower due to the influence of the crowds, which will further influence the overall traveling time. Sometimes people may be more conscious of time, so just considering the length of a path is not enough. For example, there are usually many people in the security check area in an airport, so passengers have to wait to pass it. If we just consider the length of a path, it may still result in missing a flight.

There are some challenges to solve the aforementioned problems. First, the techniques for outdoor space are not applicable to indoor space because of the complex topology in indoor space. Second, there is no efficient and effective semantic foundation for indoor space. Third, there is no indoor model or index to handle the temporal variations in indoor space. Fourth, when considering the indoor crowds, we usually have to derive the population in some partitions at a future time. There is no framework to handle this problem. To bridge the gap, we propose and study the following problems.

1) **Experimental study of techniques for indoor spatial queries.** To study the performance and differences of different techniques for indoor spatial queries, we conduct extensive experiments on existing proposals. We survey four different indoor spatial queries and five indoor model/indexes and experimentally evaluate the corresponding query processing algorithms [16].

2) **Indoor Keyword-aware Routing Query.** We study a new query called Indoor Top-k Keyword-aware Routing Query (IKRQ) [8]. It returns k routes from the given point s to the given point t within a given distance constraint. These k routes have the highest ranking scores integrating keyword relevance and spatial distance [8]. We design a fundamental model to organize textual information and spatial information. We propose a method to calculate the ranking score which considers the indoor textual information and spatial distance. We design two algorithms with several pruning rules to process IKRQ.

3) **Indoor Temporal-variation aware Routing Query.** We propose a novel query called Indoor Temporal-variation aware Shortest Path Query (ITSPQ) [15]. It

returns the valid shortest path based on the latest indoor topology at the given query time [15]. To answer an ITSPQ, the Indoor Temporal-variation Graph (IT-Graph) is proposed to organize indoor spatial information with temporal variations. Two corresponding algorithms are proposed to process the query. The main difference between the two algorithms is that one checks a door's accessibility synchronously while the other one check that asynchronously. To speed up query processing, the Indoor Temporal-variation Index (IT-Index) is designed which can maintain dynamic indoor door-to-door distances.

4) **Indoor Crowd-aware Routing Query.** We propose and study two types of crowd-aware indoor path planning queries [17]. The Indoor Crowd-Aware Fastest Path Query (`FPQ`) returns a path with the shortest travel time in the presence of crowds, whereas the Indoor Least Crowded Path Query (`LCPQ`) returns a path encountering the least objects en route [17]. We propose a unified framework that consists of three main components including an indoor crowd model, a time-evolving population estimator, and two exact and two approximate algorithms to process each type of query [17].

1.2 Thesis Structure

This thesis mainly studies the spatial queries for indoor location-based services. The overall structure is illustrated in Fig. 1.2.

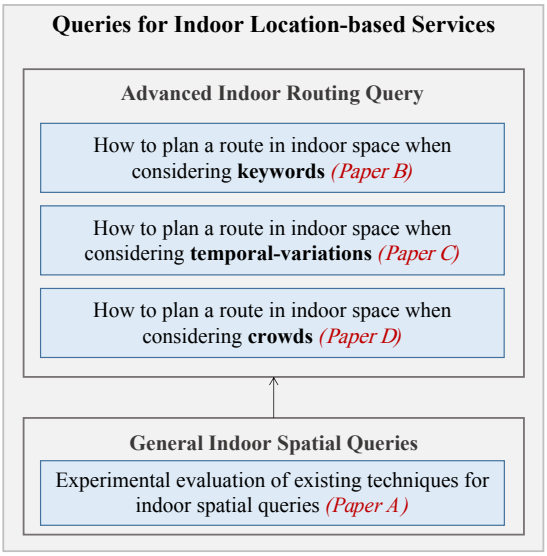


Fig. 1.2: Thesis Structure

It first analyzes the existing techniques for indoor spatial queries and conducts ex-

tensive experiments to study the performance and differences of these proposals. Then it studies three specific advanced indoor routing queries considering keywords, temporal variations, and crowds, respectively. Paper A [16] evaluates five different indoor model/indexes for four indoor spatial queries. This work is the foundation of other works. Paper B [8] studies an indoor keyword-aware routing query that considers the indoor textual information. Paper C [15] proposes an indoor temporal-variation aware shortest path query that considers temporal variations such as the door's active time. Paper D [17] studies two queries, indoor crowd-aware fastest path query and indoor least crowded path query which considers the influence of indoor crowds for indoor spatial queries.

Chapter 2

Experimental Study on Indoor Spatial Queries

This chapter gives an overall introduction of Paper A [16]. The chapter reuses content from the paper when that is considered most effective.

2.1 Motivation and Background

Many people's activities are in indoor space, so indoor location-based services (LBS) attract the attention of both industrial and academic communities [3, 5]. Typical indoor spatial queries are usually the foundation of different applications, such as POI search [13, 20] and path planning [7–9]. Several techniques like space models, indexes, and algorithms have been proposed to support different indoor queries, such as range query, kNN query, and shortest path/distance query. However, it is still hard for developers to choose an appropriate technique of indoor queries for a certain indoor environment because there is still no comprehensive experimental study on all these approaches. Therefore, it is significant to study the pros and cons of typical indoor techniques and recommend optimal methods for a given indoor scenario. In this work, we study five indoor models/indexes that are the foundation of four indoor spatial queries. We compare their technical features theoretically and analyze the complexity of all algorithms. We design a benchmark to evaluate their performance in different indoor scenarios. We report the experimental results and summarize their advantages and disadvantages.

2.2 Techniques of Indoor Spatial Query

2.2.1 Indoor Spatial Query Types

We study the following four typical indoor spatial query types. We do not consider moving objects in this work.

1) **Range Query (RQ).** A *range query* $\mathsf{RQ}(p,r)$ returns all indoor objects from object set O whose indoor distance from query point p is within distance r [16].

2) k **Nearest Neighbor Query (kNNQ).** A k *nearest neighbor query* $k\mathsf{NNQ}(p)$ returns an object set O' of k indoor objects whose indoor distances from query point p are the smallest [16].

3) **Shortest Path Query (SPQ).** A *shortest path query* $\mathsf{SPQ}(p,q)$ returns the shortest path $\phi = \langle p, d_i, \ldots, d_j, q \rangle$ from source point p to target point q [16].

4) **Shortest Distance Query (SDQ).** A *shortest distance query* $\mathsf{SDQ}(p,q)$ returns the shortest indoor distance from source point p to target point q, i.e., the length of $\mathsf{SPQ}(p,q)$ [16].

2.2.2 Indoor Models and Indexes

We compare five indoor models and indexes.

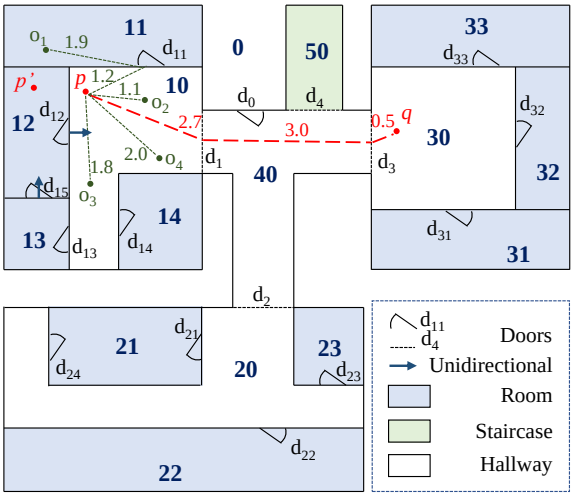
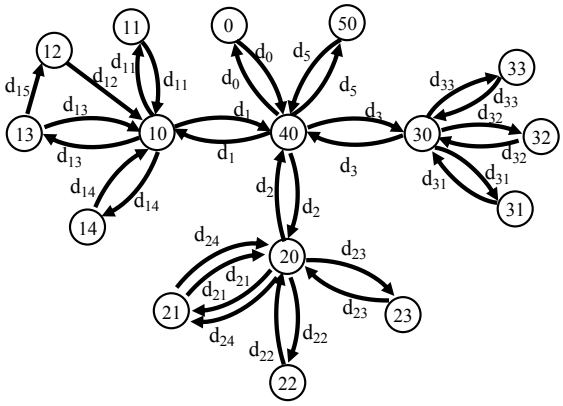


Fig. 2.1: Example Floorplan [16]

Indoor Distance-Aware Model [19] (IDMODEL). The graph-based IDMODEL $G_{\text{dist}}(V, E_a, L, f_{\text{dv}}, f_{\text{d2d}})$ consists of five tuples. V represents a set of vertexes, and each vertex is an indoor partition. E_a represents a set of edges, and each edge connects two partitions. L represents the set of edge labels, and each label maintains the information of a door. f_{dv} is a door-to-partition distance mapping for each partition. The distance

is the maximum Euclidean distance from the door to the point in the partition. f_{d2d} is a door-to-door distance mapping. If two doors are connected to the same partition, the distance is the Euclidean distance between these two doors. An example of IDMODEL corresponding to Fig. 2.1 is shown in Fig. 2.2.



Door-to-partition distance map

Key	Value
(d_0, v_{40})	*3.5m*
(d_0, v_{30})	*6.1m*
…	

Door-to-door distance map

Key	Value
(v_{40}, d_0, d_1)	*1m*
(v_{40}, d_0, d_5)	*1.2m*
…	

Fig. 2.2: An Example of IDMODEL [16]

Indoor Distance-Aware Index [19] (IDINDEX). Based on IDMODEL, the IDINDEX maintains precomputed global door-to-door distances and their ordering in two matrices to help further speed up query processing. The **door-to-door distance matrix** M_{d2d} stores all precomputed door-to-door shortest distance. The size of M_{d2d} is $\mathsf{N} \times \mathsf{N}$ where N means all number of doors in indoor space [16]. The **distance index matrix** M_{idx} is also an $\mathsf{N} \times \mathsf{N}$ 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 [16]. Fig. 2.3 illustrates an example of the IDINDEX matrices for the top-left part in Fig. 2.1 [16].

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

(a) Distance Matrix M_{d2d}

$$\begin{pmatrix} & 1 & 2 & 3 & 4 & 5 & 6 \\ d_1 & d_1 & d_{11} & d_{12} & d_{14} & d_{13} & d_{15} \\ d_{11} & d_{11} & d_1 & d_{12} & d_{14} & d_{13} & d_{15} \\ d_{12} & d_{12} & d_{15} & d_{14} & d_{11} & d_{13} & d_1 \\ d_{13} & d_{13} & d_{15} & d_{12} & d_{14} & d_1 & d_{11} \\ d_{14} & d_{14} & d_{13} & d_{12} & d_{15} & d_1 & d_{11} \\ d_{15} & d_{15} & d_{13} & d_{12} & d_{14} & d_{11} & d_1 \end{pmatrix}$$

(b) Distance Index Matrix M_{idx}

Fig. 2.3: An Example of IDINDEX [16]

Composite Indoor Index [25] (CINDEX). The CINDEX consists of three layers to maintain indoor spatial information and moving objects. The **geometric layer** organizes all indoor partitions based on R*-tree [4]. A skeleton tier is designed to han-

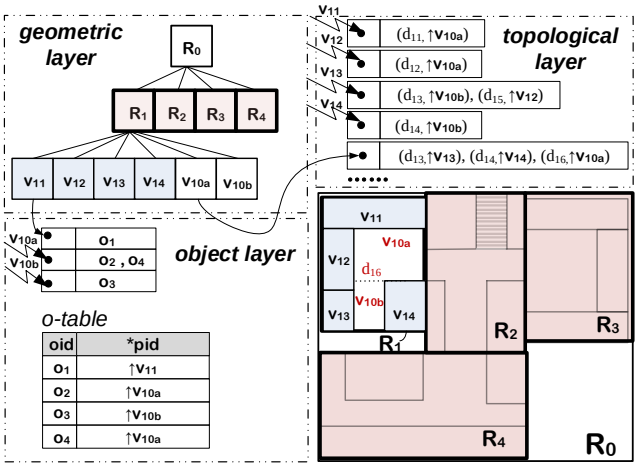
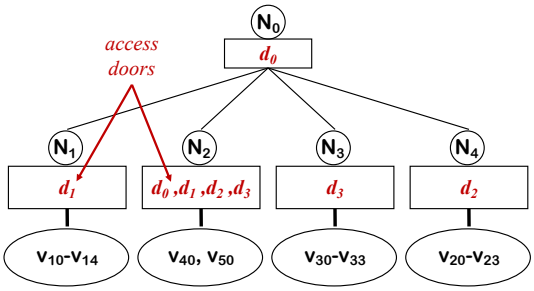


Fig. 2.4: CINDEX Example [16] (Adapted from [25])

dle the distances between staircases on different floors. The **topological layer** mainly keeps the connectivity information among indoor partitions. There are inter-partition links connecting the geometric layer and topological layer [16]. The **object layer** stores object buckets for each indoor partition at the leaf node level of the geometric layer [16]. We focus on static objects in this study, so we adapt the object layer to index static indoor objects [16]. Fig. 2.4 shows an example of CINDEX for Fig. 2.1.



	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)

Fig. 2.5: An Example of IP-TREE [16]

IP-Tree and VIP-Tree [24] (IP-TREE). The IP-TREE maintains indoor spatial information in a tree with a number of door-to-door distances matrices in each node. Specifically, several topologically adjacent indoor partitions are maintained in each leaf node [16]. The non-leaf node consists of a set of adjacent leaf nodes, and adjacent non-leaf nodes are combined hierarchically until a root node is formed [16]. There are a set of **access doors** and a **distance matrix** in each node N. An access

door connects the node N to its external space. Each leaf node contains a small distance matrix that stores the shortest distance between each door in this leaf node to each access door of the leaf node [16]. Each non-leaf node also maintains a distance matrix that keeps the shortest distances between each pair of access doors of its child nodes [16]. Fig. 2.5 illustrates an example of IP-TREE for Fig. 2.1. Vivid IP-Tree (VIP-TREE) [24] maintains more precomputed information in the tree to further speed up the distance computation.

2.2.3 Comparison

We compare the feature, complexity, and extensibility of five indoor model/indexes in Tables 2.1, 2.2, and 2.3.

Table 2.1: Feature Comparison [16]

Models	IDMODEL	IDINDEX	CINDEX	IP-TREE	VIP-TREE
Precompute	No	Yes	No	Yes	Yes
Structure	Graph+Mappings	Matrix	Tree+Links	Tree+Matrix	Tree+Matrix
Initialization	Sequential scan	Sequential scan	R*-Tree pruning	Sequential scan	Sequential scan
Expansion	Dijkstra	Loop	Dijkstra	LCA	LCA
RQ	△	✓	✓	✓	✓
*k*NNQ	△	✓	✓	✓	✓
SPQ	✓	△	△	✓	✓
SDQ	✓	△	△	✓	✓

2.3 Experimental Evaluation

2.3.1 Benchmark

Datasets. We use four various indoor space datasets, i.e., SYN, MZB, HSM, and CPH, each featuring a different indoor space [16]. Fig. 2.6 shows the floorplans of these four datasets.

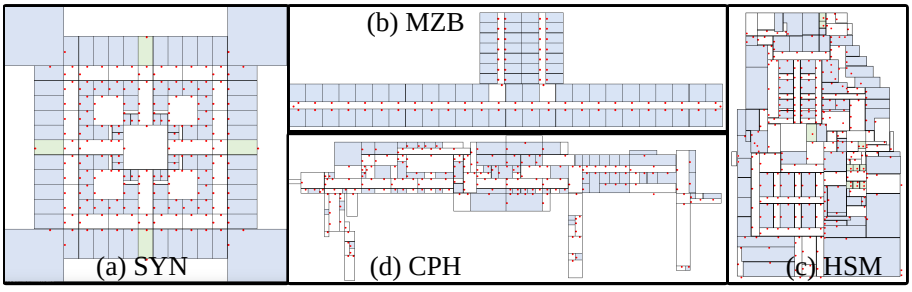


Fig. 2.6: Floorplan of Datasets [16]

Table 2.2: Complexity Analysis [16]

	Space	RQ	*k*NNQ	SDQ	SPQ
IDMODEL	$\mathscr{O}(V+D+2Vd+Vd^2)$	$\mathscr{O}(oV\log D)$	$\mathscr{O}(oV\log D)$	$\mathscr{O}(V\log D)$	$\mathscr{O}(V\log D+w)$
IDINDEX	$\mathscr{O}(2D^2)$	$\mathscr{O}(od\log D)$	$\mathscr{O}(od\log D)$	$\mathscr{O}(d^2)$	$\mathscr{O}(d^2+w)$
CINDEX	$\mathscr{O}(V+Vd+O)$	$\mathscr{O}(oV\log D)$	$\mathscr{O}(oV\log D)$	$\mathscr{O}(V\log D)$	$\mathscr{O}(V\log D+w)$
IP-TREE	$\mathscr{O}(\rho^2 f^2 L+\rho D)$	$\mathscr{O}((\rho\log_f L)^2(Vo/L+\rho))$	$\mathscr{O}((\rho\log_f L)^2(Vo/L+\rho))$	$\mathscr{O}(\rho^2\log_f L)$	$\mathscr{O}((\rho^2+w)\log_f L)$
VIP-TREE	$\mathscr{O}(\rho^2 f^2 L+\rho D\log_f L)$	$\mathscr{O}(\rho^2\log_f L(Vo/L+\rho))$	$\mathscr{O}(\rho^2\log_f L(Vo/L+\rho))$	$\mathscr{O}(\rho^2)$	$\mathscr{O}(\rho^2+w)$

Table 2.3: Extensibility Analysis [16]

	IDMODEL	IDINDEX	CINDEX	IP/VIP-TREE
Temporal Variation	✓	X	✓	X
Moving Objects	✓	✓	✓	✓
Uncertain Locations	X	X	✓	X
Keywords	✓	✓	✓	✓

Performance Evaluation Procedure [16]. We propose a number of tasks to evaluate the performance of each model/index. These tasks are classified into two types, i.e., A. model construction and B. query processing. For each task, we vary one parameter while others are fixed to the default value. The experimental settings with default parameters in bold are listed in Table 2.4. The code of the evaluations is public online [1]. The following tasks are reproduced from [16].

A **Model Construction.** We use (**a1**) model/index size and (**a2**) construction time as the performance metrics of model construction. We evaluate the effect of the indoor size by varying the number of floors from 3 to 9 in the synthetic dataset.

B **Query Processing.** We use (**b1**) running time and (**b2**) memory use as the performance metrics to evaluate the efficiency of query processing (all query types). We use (**b3**) number of visited doors (NVD) to evaluate SPDQ [16].

B1 **Effect of Floor Number n.** For each query type, we vary floor number n from 3 to 9 in SYN to evaluate the scalability [16].

B2 **Effect of Object Number $|O|$.** We vary $|O|$ from 500 to 2500 to evaluate the efficiency of RQ and kNNQ in four datasets [16].

B3 **Effect of Range Distance r.** We increase r from 200m to 1000m in SYN5, HZM and CPH, and from 20m to 100m in MZB to evaluate the effect of r to RQ [16].

B4 **Effect of k.** For kNNQ, we increase k from 1 to 100 in four datasets [16].

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 [16].

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$^+$ [16].

B7 **Effect of Hallway's Decomposition Method.** We use SYN5 and MZB with the derived datasets, SYN5^0, MZB^0 and MZB$^\Delta$ [16].

2.3.2 Performance Analysis

We conduct extensive experiments, then report and analyze the results. We implement all techniques and experiments in Java and run on a MAC with a 2.30GHz Intel i5 CPU and 16 GB memory [16].

The performance of five model/indexes is summarized in Table 2.5 where more stars mean it performs better for the corresponding query type [16].

Table 2.4: Evaluation Settings (Default Parameters in Bold) [16]

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$^\Delta$

- IDMODEL costs minimum time and space for construction. For SPQ/SDQ, it works better if large paritions are decomposed into more small partitions [16]. It always performs well for RQ and kNNQ.
- IDINDEX performs well for all query types regarding the time costs. However, it incurs large time and space costs when constructing offline.
- CINDEX's performance of query processing is similar to IDMODEL [16].
- IP-TREE and VIP-TREE performs well for SPQ/SDQ tasks, especially when the indoor space contains more C-Pars connected by so-called access doors [16].

In a word, IDINDEX is the best choice when the size of doors and partitions is small. VIP-TREE is recommended for path planning or the space accommodates many C-Pars [16]. Otherwise, IDMODEL is a good choice for RQ and kNNQ because it requires small time and space to construct and it can balance storage and query time costs well [16].

Table 2.5: Summary of Findings [16]

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

Chapter 3

Indoor Keyword-aware Routing Query

This chapter gives an overall introduction of Paper B [8]. The chapter reuses content from the paper when that is considered most effective.

3.1 Problem Motivation and Statement

People usually spend significant parts of their daily life in different indoor venues such as airports, office buildings, and shopping malls. The demand for keyword-aware routing queries is increasing, especially when people have to go through an unfamiliar large indoor building [8]. For example, Jesper wants to take a flight in Copenhagen Airport. Before he reaches the boarding gate, he wants to draw some cash at an ATM, have lunch, and buy some Danish cookies as a gift for friends. He has to do these within 1.5 hours. In this case, he needs an appropriate route, which is from his current position to the boarding gate and has to include an ATM, a restaurant, and a cookies shop. Besides, the distance of the route should be less than a distance constraint [8].[1]

We propose a novel query, indoor top-k keyword-aware routing query (IKRQ) [8]. An IKRQ requires a start point s, a terminal point t, a distance constraint Δ, and a query keyword list QW [8]. It returns the k best routes from s to t that are not longer than Δ and have highest ranking scores [8]. A route score integrates the route's keyword relevance w.r.t. QW and its route distance, i.e., length from s to t [8].The formal definition of our problem is defined as follows, which is reproduced from [8].

Problem (Indoor Top-k Keyword-aware Routing Query [8]). *Given a start point p_s, a terminal point p_t, a distance constraint Δ, and a query keyword list QW, an* **indoor top-k keyword-aware routing query** *$IKRQ(p_s, p_t, \Delta, QW, k)$ returns k*

[1] A time constraint T, e.g., 1.5 hours, can easily be converted to a distance constraint $\Delta = V_{max} \cdot T$, where V_{max} is the maximum indoor walking speed [8].

routes from p_s to p_t in a k-set Θ such that $\forall R \in \Theta$, $\delta(R) \leq \Delta$ and $\Psi(R,\Delta,QW) \geq \Psi(R',\Delta,QW)$ for any route $R' \notin \Theta$ from p_s to p_t with $\delta(R') \leq \Delta$.

Above, $\delta(R)$ represents the indoor distance of a route R. $\Psi(R,\Delta,QW)$ captures the ranking score for a route R, which takes into account both spatial distance and keyword relevance between R and a given routing query. We proceed to detail the design of our ranking mechanism for routes [8].

3.2 Ranking Relevant Routes

3.2.1 Spatial Distance

In indoor space, a ***route*** $R = (x_s, d_i, \ldots, d_n, x_t)$ is a path through a sequence of doors from an item x_s to an item x_t, where x_s and x_t can be a point or a door [8]. A route R's ***route distance*** is $\delta(R) = \delta_*(x_s, d_i) + \sum_{k=i}^{n-1} \delta_*(d_k, d_{k+1}) + \delta_*(d_n, x_t)$, where $\delta_*(x_1, x_2)$ means the Euclidean distance from point/door x_1 to point/door x_2. If x_1 and x_2 are not in a same partition, $\delta_*(x_1, x_2)$ is 0 [8].

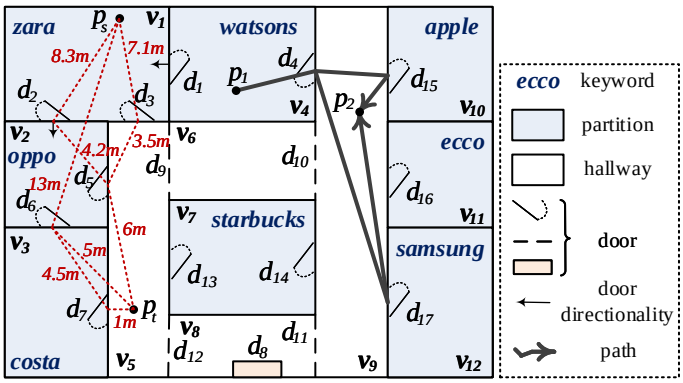


Fig. 3.1: An Example of Floorplan [8]. ©2020 IEEE

Referring to Fig. 3.1, for route $R(p_s, d_2, d_5, p_t)$, the route distance $\delta(R) = \delta_*(p_s, d_2) + \delta_*(d_2, d_5) + \delta_*(d_5, p_t) = 8.3m + 4.2m + 6m = 18.5m$.

For a partition, which covers the start point p_s, the terminal point p_t, or a subset of query keywords, we call it **key partition** [8]. The set of sequential key partitions for a route is denoted as $KP(\cdot)$ [8].

We call routes R_i and R_j ***homogeneous routes*** if $R_i.head = R_j.head$, $R_i.tail = R_j.tail$, and $KP(R_i) = KP(R_j)$ [8]. Suppose HR is a complete set of homogeneous routes for a routing query, we say a route $R_i \in HR$ is ***prime*** against $R_j \in HR$ if $\delta(R_i) < \delta(R_j)$ [8]. R_i is a ***prime route*** if R_i is prime against all other routes in HR [8].

3.2.2 Keyword Relevance

We consider two keyword types, i.e., **identity word** (i-word) and **thematic word**(t-word) [8]. We call an indoor partition's semantic name as an i-word, while we call a tag relevant to that partition as a t-word [8]. In particular, i-words are usually obtained from the floor map. For example, in a shopping mall, i-words can be store names like *costa* and *samsung* or function area names such as *toilet* and *elevator*. T-words are usually extracted from the textaul information of the indoor partition. For instance, a shop *costa* can be associated with many t-words such as *coffee*, *tea* and *milk*. In our setting, one partition can associate with many t-words but only one i-word [8]. We use a **P2I** mapping $P2I(v_k)$ to map a partition v_k to its associated i-word w_i, and an **I2P** mapping $I2P(w_i)$ to map an i-word w_i to a set of relevant partitions [8]. Moreover, We use an **I2T** mapping $I2T(w_i)$ to map an i-word w_i to a set of relevant t-words, and a **T2I** mapping $T2I(w_t)$ to map a t-word w_t to a set of relevant i-words [8]. For each partition v_k, we use $\{P2I(v_k), I2T(P2I(v_k))\}$ to describe its **partition words** $PW(v_k)$, which includs an i-word $w_i = P2I(v_i)$ and a set of t-words relevant to w_i as indicated by I2T mapping [8]. Fig. 3.2 shows an example of keyword mapping. In our setting, we assume that the i-word set and t-word set are distinct. Moreover, for two partitions with the same i-word, we assume that they have the same set of t-words [8].

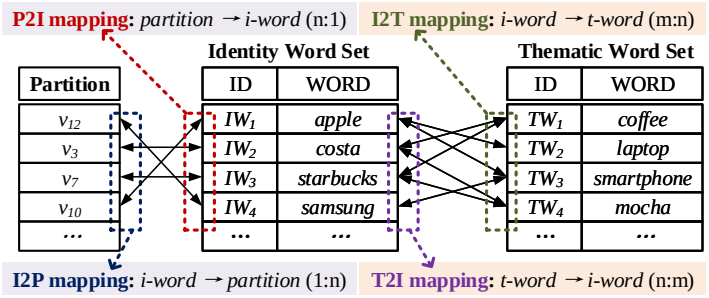


Fig. 3.2: Indoor Space Keyword Mappings [8]. ©2020 IEEE

For a route $R = (x_s, d_i, \ldots, d_n, x_t)$, its ***route words*** are the union of all its relevant partitions' associated i-words, computed as $RW(R) = \bigcup_{x \in R} PW(v_*(x)).w_i$ [8]. Referring to the route $R = (p_s, d_3, p_t)$ in Fig. 3.1, for point p_s, we have $PW(v(p_s)).w_i = PW(v_1).w_i = \{zara\}$. Likewise, we have $PW(v(p_t)).w_i = \varnothing$ [8]. For door d_3, we have $PW(D2P_{\sqsubset}(d_3)).w_i = PW(v_1).w_i \cup PW(v_5).w_i = \{zara\} \cup \varnothing = \{zara\}$ [8]. Consequently, we have $RW(R) = \{zara\}$ [8].

Given a query keyword list QW, we define the route R's keyword relevance as follows, which reproduced from [8].

$$\rho_{QW}(R) = \begin{cases} 0, & \text{if } N_{QW}(R) = 0; \\ N_{QW}(R) + \dfrac{\sum_{w_Q \in QW} \left(\max_{w'_i \in \mathrm{M}(w_Q, R)} s(w'_i) \right)}{N_{QW}(R)}, & \text{otherwise.} \end{cases}$$

Above, $N_{QW}(R)$ is the number of R's route words that are relevant to query words in QW, and $\mathrm{M}(w_Q, R)$ denotes the set of w_Q's matching i-words on R [8]. When the context is clear, we use $\rho(R)$ to denote the keyword relevance [8].

3.2.3 Ranking Score for Routes

Given a query IKRQ(p_s, p_t, Δ, QW, k) and a route R from p_s to p_t, we compute the **ranking score** of R as follows [8].

$$\psi(R, \Delta, QW) = \alpha \cdot \frac{\rho(R)}{|QW|+1} + (1-\alpha) \cdot \left(\frac{\Delta - \delta(R)}{\Delta}\right) \tag{3.1}$$

The ranking score is a linear combination of keyword relevance and spatial relevance, where the tradeoff parameter $\alpha \in [0,1]$ is used to customize the weight of two parts [8].

3.3 Query Processing Algorithms

We design a unique processing framework for IKRQ, which is shown in Fig. 3.3, which is reproduced from [8].

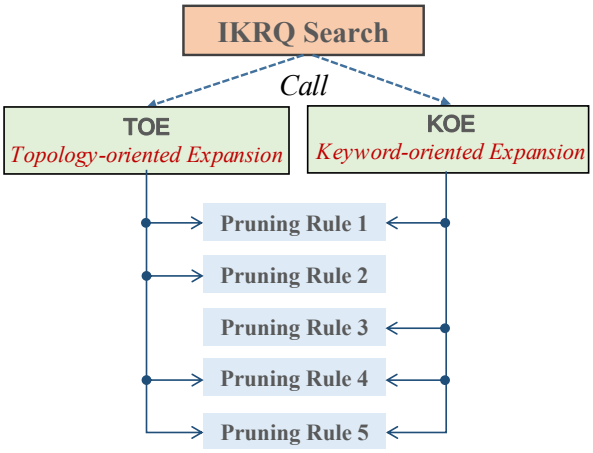


Fig. 3.3: IKRQ Processing Framework [8].

We propose two expansion methods, i.e., topology-oriented expansion (ToE) and keyword-oriented expansion (KoE) [8]. To improve the efficiency of query processing, we design 5 pruning rules.

3.3.1 Pruning Rules for Expansion

We calculate the *lower bound indoor distance* for two indoor items x_i and x_j as follows, which are reproduced from [8].

$$|x_i,x_j|_L = \begin{cases} |x_i,x_j|_E, & \text{if } x_i \text{ and } x_j \text{ are on the same floor;} \\ \min\limits_{sd_i \in SD(x_i), sd_j \in SD(x_j)} \big(|x_i,sd_i|_E + \delta_{s2s}(sd_i,sd_j) + \\ |sd_j,x_j|_E\big), & \text{otherwise.} \end{cases}$$

This method is originate from a previous work [25]. In this equation, $|x_i,x_j|_L$ is the Euclidean distance if x_i and x_j are on the same floor. Otherwise, one needs to go through a number of staircase doors (e.g., $sd_i \in SD(x_i)$) to reach x_j from x_i, and there can be multiple such paths. In this case, $|x_i,x_j|_L$ is the shortest path distance among all such paths [8].

Based on the lower bound indoor distance, we propose the following pruning rules for IKRQ [8].

Pruning Rule 1. *A partial route $R^\star = (p_s,d_i,\ldots,d_n)$ in the searching can be pruned if $\delta(R^\star) + |d_n,p_t|_L > \Delta$. [8]*

Pruning Rule 2. *A door d_n can be pruned out of the search if $|p_s,d_n|_L + |d_n,p_t|_L > \Delta$ [8].*

Pruning Rule 3. *An indoor partition v_i can be pruned out of the search if its lower bound distance $\delta(p_s,v_i,p_t)$ =*

$$\min_{d_i \in P2D_\sqsupset(v_i), d_j \in P2D_\sqsubset(v_i)} (|p_s,d_i|_L + \delta_{d2d}(d_i,d_j) + |d_j,p_t|_L) > \Delta.$$

Pruning Rule 4. *Given the current k-th highest ranking score ψ_k among the seen complete routes, a partial route $R^\star = (p_s,d_i,\ldots,d_n)$ can be pruned if its upper bound ranking score $\psi_U(R^\star) = \alpha \cdot 1 + (1-\alpha)(1 - (\delta(R^\star) + |d_n,p_t|_L)/\Delta) \leq \psi_k$ [8].*

Pruning Rule 5. *A partial route $R^\star = (p_s,d_i,\ldots,d_n)$ in the search can be pruned if the search has already obtained a route $R^{\star'}$ from p_s to d_n that is prime against $R^\star$ [8].*

3.3.2 Query Processing Algorithms for IKRQ

We design a unique query processing algorithm that can adapt to two expansion methods. Both two expansion methods are based on the spirit of Dijkstra's algorithm when expanding in an indoor graph.

Topology-oriented Expansion (ToE). The main idea of ToE is that it always expands one hop (an enterable door or the terminal point) from the current node (a leavable door or the start point). All incomplete routes are maintained in a priority queue. In each iteration, the route with a higher ranking score will be expanded first.

Keyword-oriented Expansion (KoE). KoE mainly focuses on expanding to an enterable door of the partition which is related to an uncovered keyword. One hop in KoE is actually a partial route that may go through several partitions.

3.4 Experimental Evaluation

3.4.1 Comparable Methods

We evaluate ToE, KoE, and their variants on synthetic and real data. All compared algorithms are listed in Table 3.1.

Table 3.1: Notations of Comparable Methods [8]. ©2020 IEEE

Modification	ToE family	KoE family
–	ToE	KoE
no distance-based Pruning Rules 6-8	ToE\D	KoE\D
no kbound-based Pruning Rule 9	ToE\B	KoE\B
no prime-based Pruning Rule 10	ToE\P	–
with precomputed shortest routes	–	KoE*

3.4.2 Datasets and Settings

Synthetic Dataset. We generate a multi-floor indoor space based on an existing floorplan[2]. There are 141 partitions and 220 doors on each floor. To evaluate the effect of the indoor size, we duplicate floorplan 3, 5, 7, or 9 times [8]. In the default setting, we use a 5-floor indoor space with 705 partitions and 1100 doors [8]. We obtain keywords by extracting words from the shop information of five shopping malls in Hong Kong. Finally, we get 1120 i-words and 9195 t-words. Then we randomly assign these words to each room in the indoor space [8].

Real Dataset. We use the real indoor topology of a seven-floor shopping mall in Hangzhou, China [8]. There are ten staircases in which each stairway is roughly 20m long [8]. There are 639 stores in total. We extract 5036 t-words for 533 i-words from the mall's website [8].

Parameters and Metrics. We list the parameter settings in Table 3.2, where the bold numbers are default values[3]. We generate 10 query instances for each parameter setting and run each instance five times. The metrics we use includes the *average running time* and *average memory cost* [8].

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

[3]In the real dataset, we adjust α to 0.7 to suit the needs of keyword-awareness in shopping [8].

Table 3.2: Parameter Settings [8]. ©2020 IEEE

Parameters	Settings
k	1, ..., **7**, ..., 11
$\|QW\|$	1, 2, 3, **4**, 5
β (% of i-words in QW)	20%, 40%, **60%**, 80%, 100%
δ_{s2t} (meter)	1100, 1300, **1500**, ..., 2100
η	1.4, **1.6**, 1.8, 2.0
α	0.1, 0.3, **0.5**, 0.7, 0.9
τ	0.05, **0.1**, 0.2, 0.4

3.4.3 Performance Analysis

We report the running time of each algorithm with the default setting in Fig. 3.4. With the pruning rules, ToE and KoE run fast compared to other methods. Specifically, ToE returns top-7 results within 117ms while KoE needs about 133ms [8].

In general, ToE\D and KoE\D cost more time, which indicates that the distance-based pruning is useful. Next, ToE\B and KoE\B perform similarly to their original counterparts, which means that the $k$bound pruning barely works in the default setting [8].

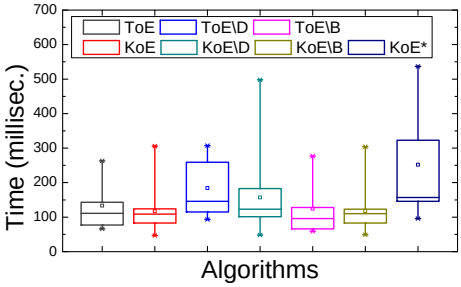


Fig. 3.4: Default parameters [8]. ©2020 IEEE

Moreover, the results in Fig. 3.4 show that KoE* incurs more time costs than other algorithms and it fluctuates more on different query instances [8]. This shows that its precomputing does not pay off. On the contrary, it requires more time to compute the path because it has to recompute indoor distances when a route regularity check fails [8]. ToE\P does not perform well, nearly 5 to 6 orders of magnitude slower than other methods, so we omit the results of ToE\P in Fig. 3.4 [8]. ToE\P causes the number of routes increasing exponentially because it searches without prime route-based pruning [8].

More performance analysis about the effect of different parameters can be seen in the full research work [8].

Chapter 4

Indoor Temporal-variation aware Routing Query

This chapter gives an overall introduction of Paper C [15] and Paper C is the extended version of a previous work [14]. The chapter reuses content from the paper when that is considered most effective.

4.1 Problem Motivation and Statement

Indoor routing queries can facilitate people in need and have attracted some researchers' interest. However, the existing techniques [19, 23] do not consider temporal variations and restrictions in indoor venues. For example, in a shopping mall, some doors may close in the evening which causes different indoor topology. Some indoor partitions like private offices in an office building will also result in different indoor topologies because we usually cannot pass through this kind of partitions in reality.

To handle these temporal variations and restrictions, we differentiate two kinds of indoor partitions, i.e., **private partitions** and **public partitions**. A private partition cannot be used in routing while a public partition can. Accordingly, we also distinguish two types of doors. A **public door** connects two public partitions, while a **private door** connects at least one private partitions. In our setting, we define the **active time interval** (ATI) of a door as [open-time, close-time). For example, [9:30, 17:10) means a door is open at 9:30 and closed at 17:10 [15].

We propose and study a novel query called *indoor temporal-variation aware shortest path query* (ITSPQ) which returns the shortest path from a source p_s to a target p_t while considering temporal variations and semantic restrictions [15]. However, such queries are not easy to handle. First, the existing techniques of modeling the indoor space do not take into account the temporal variations. Second, to speed up the query processing, the door-to-door distances are precomputed and maintained in a matrix or

a tree. However, the precomputed information will become invalid when considering the temporal variations.

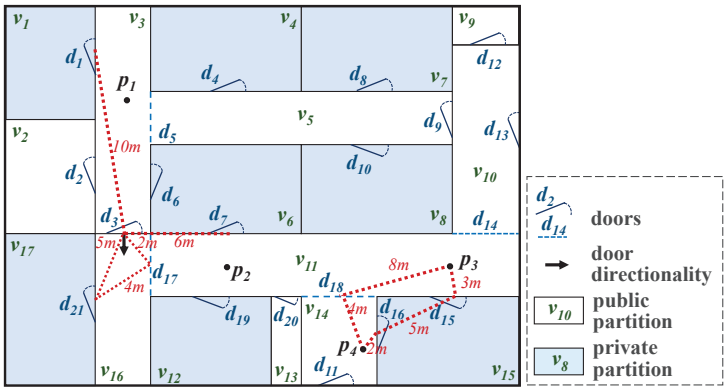


Fig. 4.1: An Example of Indoor FloorPlan [15]. ©2021 IEEE

The following problem defination and example are reproduced from [15].

Problem (Indoor Temporal-variation aware Shortest Path Query). *Given a static start point p_s, a static target point p_t, and a current timestamp t, an* indoor temporal-variation aware shortest path query ITSPQ*(p_s, p_t, t) returns the valid shortest path from p_s to p_t that meets the following rules:*

1. *Each door d_i in the path should be open at $t + \Delta t$[1], where Δt is the walking time from p_s to d_i and it is computed based on human's average walking speed [2] — $5km/h$;*

2. *The path should not go through any private partition except the private partitions that contain p_s and/or p_t.*

Referring to Fig. 4.1, for a query ITSPQ(p_3, p_4, 9:00), there are two candidate indoor paths. One path is $(p_3, d_{15}, d_{16}, p_4)$ with length $10m$ and another path is (p_3, d_{18}, p_4) with length $12m$. Although (p_3, d_{18}, p_4) is the longer one, the query returns (p_3, d_{18}, p_4) as the result. That is because $(p_3, d_{15}, d_{16}, p_4)$ breaks the rule, i.e., the path goes through a private partition v_{15}. When considering another query ITSPQ(p_3, p_4, 23:30), there is no result because d_{18} is closed at 23:00 and no path can meet both rules in the problem definition [15].

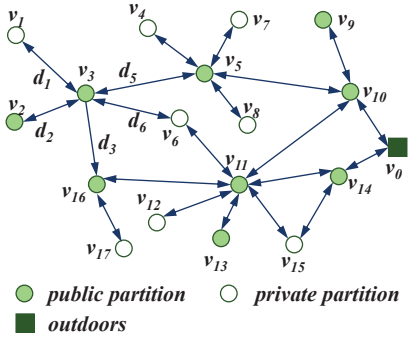
4.2 ITSPQ using Temporal-variation Graph

[1] In this paper, we do not consider the waiting tolerance in the routing, i.e., someone reaches a door and waits there until the door opens [15].

4.2.1 Indoor Temporal-Variation Graph

We design IT-GRAPH G_{IT} (V, E, L_v, L_E) which manages several information, i.e., indoor topology, semantic properties of indoor entities, geometric information, and temporal variation information in a composite structure [15]. The IT-GRAPH consists of four components. 1) The set of vertices V represents indoor partitions. 2) The set of directed edges E which means doors. 3) The set of vertex labels L_v where each label maintains three kinds of information, i.e., partition's ID, partition's type (public partition PBP and private partition PRP), and distance matrix in that partition. 4) The set of edge labels that maintains the door's ID, door type (public door PBD and private door PRD), and the ATIs of doors [15].

We illustrate an example of the IT-GRAPH in Fig. 4.2, which is reproduced from our full research paper [15]. We use solid circular vertices and hollow circular vertices to denote public and private partitions, respectively. The outdoor space is represented by a square vertex. Each directed edge represents the door, which is connected to two partitions. The information of the vertex labels and the edge labels is maintained in a door table and a partition table, respectively. For example, a record (d_1, *PRD*, ⟨[5:00, 23:00)⟩) in Fig. 4.2 means door d_3's type is public and its active time interval is [6:00, 23:00). Moreover, it shows partition v_{16} is a public partition, and the distance between its doors d_3 and d_{21} is 4m in partition table in Fig. 4.2.



Door Table

ID_d	*d-type*	*ATIs*
d_7	PRD	⟨[6:00, 23:30)⟩
d_3	PBD	⟨[6:00, 23:00)⟩
...	...	...

Partition Table

ID_v	*p-type*	*DM*
v_1	PRP	[[(d_1, d_1), 0]]
v_{16}	PBP	[[(d_3, d_{17}), 2], [(d_3, d_{21}), 4], [(d_{17}, d_{21}), 5]]
...	...	...

Fig. 4.2: Example of Indoor Temporal-Variation Graph [15]. ©2021 IEEE

4.2.2 IT-GRAPH based ITSPQ Processing

We design a unique framework to process ITSPQ using IT-GRAPH. The main idea is to expand the node in IT-GRAPH based on Dijkstra's algorithm. Due to some temporal variations and restrictions, some checks are needed to make sure whether the door is open or the partition is accessible. We propose two checking methods, synchronous check, and asynchronous check.

Synchronous Check. For searching a feasible path, we calculate the arrival time to a door d by using the departure time plus the travel time cost ($dist/velocity$). Then,

we check d's active time intervals. If the arrival time is not in d's active time intervals, we do not consider d in that path.

Asynchronous Check. One of the drawbacks of the synchronous check is that it needs to validate each encountered door when searching a route. However, in some circumstances, the graph can be static for a time interval. Because the temporal variation of doors can only happen at the border of doors' ATIs. Such border time points are called *checkpoints*. The indoor topology will stay static between two consecutive checkpoints [15]. Therefore, we propose the asynchronous check, which just checks on a time-dependent IT-GRAPH that contains all currently open doors [15]. We just need to update the topology asynchronously at each checkpoint [15].

These two check methods are suitable for different scenarios according to their characteristics. For some scenarios with improvised variations, the search using the synchronous check is recommended. For example, if a fire happens in an indoor space, some doors should be closed urgently. In this case, the indoor topology is dynamic. In contrast, for the scenario where doors are opened and closed periodically, an asynchronous check is more suitable because it can save more search costs without on-the-fly handling of active time intervals [15].

4.3 ITSPQ using Temporal-variation Index

4.3.1 Indoor Temporal-Variation Index

To speed up the query processing, we design an index, **indoor temporal-variation index** (IT-INDEX). The IT-INDEX captures the information of the indoor topology and organizes indoor partitions into a tree structure [15]. However, the tree structure does not include the temporal variations and directionality of doors, so we design a distance cube to maintain the door-to-door distance considering the temporal variations [15].

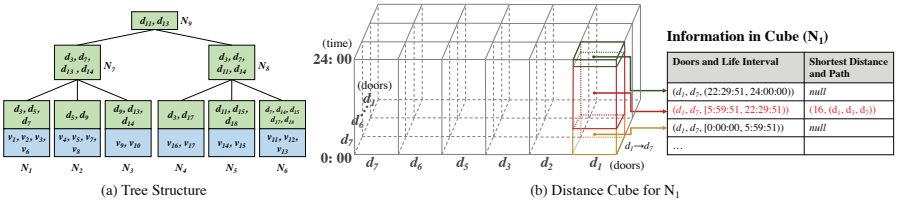


Fig. 4.3: Indoor Temporal-Variation Index [15]. ©2021 IEEE

We consider two partition types, i.e., **impassable partitions** that include all private partitions and those public partitions with only one door and **passable partitions** that are public partitions with two or more doors [15]. Each **leaf node** consists of a set of topologically interconnected partitions, while each **non-leaf node** is formed with a set of interconnected leaf nodes [15]. The non-leaf nodes are hierarchically merged to form a non-leaf node at a higher level until one root node at the highest level is

formed [15]. Corresponding to Fig. 4.1, the tree structure of IT-INDEX is illustrated in Fig. 4.3(a). Following the IP-tree [24], we maintain *access doors* in each leaf node N_i. Different from IP-tree, we distinguish **enterable access doors** and **leavable access doors** for N_i to maintain the information of door directionality [15]. For each non-leaf node, we maintain the pointers for its access doors.

When constructing an IT-INDEX, it follows two rules [15]. 1) There is at least one passable partition in a leaf node, and the other partition should be connected to the passable partition. That can guarantee that each partition in the node can be physically reachable (without considering temporal variations and door directionality) [15]. 2) The number of passable partitions in a leaf node with more than k public doors should be only one. This is to reduce the complexity of further distance computation.

For each node in the IT-INDEX, we maintain a *distance cube*, denoted as a 3-tuple (d_i, d_j, L), to store the shortest path relevant to that node [15]. In the distance cube, d_i,d_j are two doors, and L is a life interval during which the shortest path is valid (considering temporal variation) [15]. The shortest path here conforms to our rule that one can not pass any private door [15]. Compared to IP-tree [24], IT-INDEX maintains semantic properties and temporal variations of indoor entities, together with the distance cube that keeps the shortest distance information with respect to temporal variations [15].

4.3.2 IT-INDEX based ITSPQ Processing

We propose an overall framework to process ITSPQ using IT-INDEX. It first finds thee *lowest common ancestor* N_{LCA} for the points p_s and p_t [15]. Then, two children of N_{LCA} are obtained, i.e., one is the ancestor of $\texttt{Leaf}(p_s)$, N_s, while another is the ancestor of $\texttt{Leaf}(p_t)$, N_t. Next, it computes the shortest paths from p_s to each leavable access door of N_s and finds qualified leavable access doors of N_s. For each qualified d_i, it computes the shortest distance from d_i to each enterable access door of N_t [15]. Then, it computes the shortest distance from d_j to p_t for each qualified d_j. Consequently, the shortest distance *dist* from p_s to p_t through d_i and d_j is computed as the sum of the shortest distances of $p_s \rightarrow d_i$, $d_i \rightarrow d_j$ and $d_j \rightarrow p_t$ [15].

4.4 Experimantal Evaluation

4.4.1 Datasets and Settings

Synthetic Dataset. We generate a multi-floor indoor space with 141 partitions and 224 doors on each floor [15]. In our setting, each room partition is seen as a private partition (PRP) while each common partition such as a hallway or a staircase is seen as a public partition (PBP). Finally, there are 53 PBPs and 88 PRPs on each floor. We use a 7-floor indoor space with 987 partitions (371 PBPs and 616 PRPs) and 1568 doors as the default setting [15].

Real Dataset. We collect a dataset with real indoor topology and temporal variation information from a seven-floor shopping mall in Hangzhou, China [15]. All stores and equipment rooms are seen as private partitions while hallways and staircases are seen as public partitions [15]. Totally, there are 1050 partitions (553 PBPs and 497 PRBs) connected by 2093 doors.

Parameter Settings. The parameter settings with default values in bold are listed in Table 4.1. The floor number is only varied in synthetic data to evaluate the scalability. We select random pairs of open time and close time to form the checkpoint set T in size of 4, **8**, 12, or 16 [15]. T is set in the default value 8 in real data. We set a temporal door ratio (TDR) (20%, 40%, **60**%, 80% or 100%) of doors to be the *varied doors* that do not open all time [15]. TDR is set in the default value 60% in real data [15]. For each setting of s2t, we generate five pairs of p_s and p_t to form the query instances [15]. The distance from p_s to p_t is varied from 1100m to 1900m. We fix the query time to 12:00 in each query instance to make a fair comparison. We also vary the query time t to see its effect on query processing [15].

Performance Metrics. We evaluate the construction time and index size of IT-INDEX. For query processing algorithms, we evaluate the *average* running time, memory cost, and the number of door visits (NDV) per run of a single query instance [15]. To obtain the average value, each query instance is run 10 times.

Table 4.1: Parameter Settings for Synthetic Data [15]. ©2021 IEEE

Parameters	Settings
Floor Number	3, 5, **7**, 9, 11
$\|T\|$	4, **8**, 12, 16
TDR (% of varied doors)	20%, 40%, **60%**, 80%, 100%
s2t (m)	1100, 1300, **1500**, 1700, 1900
t	0:00, 2:00, ..., **12:00**, ..., 22:00

Baseline Method [15]. We compare our proposals to a general temporal graph (GTG) [10, 11, 21, 22]. To adapt to our problem, we assign the door type and ATIs to each vertex in GTG and the weight of each edge is the distance between two doors. We adapt the synchronous check to GTG. We do not consider door directionality in the comparative experiments because it will cost more space and search time for GTG [15].

4.4.2 Performance Analysis

For synthetic dataset, in the default parameter setting, it takes around 310 ms to build the IT-GRAPH and the graph size is around 3.5 MB, while the constructing time for IT-INDEX is around 30 minutes and its size is around 7 MB [15]. Calculating the distance cubes for nodes in IT-INDEX takes the most of the time when constructing IT-INDEX. For real dataset, IT-INDEX is built within 4 hours and its size is around 14.3 MB [15].

We summarize the results of experiments on query processing as follows [15]. ITI always performs best in terms of efficiency because IT-INDEX maintains some door-to-door distances in each node, which can help speed up searching. The running time of using ITI is shorter than the other three by an order of magnitude in most tests. If the graph topology is more completed, the efficiency of ITG/S, ITG/A and GTG will become worse dramatically, whereas ITI's running time and memory use just increase slightly. Besides, GTG has the lowest search efficiency because of its large graph size [15].

Chapter 5

Indoor Crowd-aware Routing Query

This chapter gives an overall introduction of Paper C [17]. The chapter reuses content from the paper when that is considered most effective.

5.1 Problem Motivation and Statement

Indoor venues like shopping malls or airports accommodate many objects. These objects may form crowds and influence how people choose an appropriate route. For instance, people's moving speed will become slower influenced by crowds, which will further influence the overall traveling time. Sometimes people are sensitive to travel time, so just considering path length is not enough. For example, in an airport, if we just consider the shortest path, it may still cause missing a flight because the crowds may influence the traveling time [17]. In other scenarios, people may want to find a path that can meet the least number of people. For instance, people may want to avoid human contact as much as possible during the COVID-19 pandemic [17].

In this work, we propose two crowd-aware indoor path planning queries [17]. Referring to Fig. 5.1, given a source point p_s, a target point p_t, and a query time t, an Indoor Crowd-Aware Fastest Path Query (`FPQ`) returns a path with the shortest travel time in the presence of crowds, whereas an Indoor Least Crowded Path Query (`LCPQ`) returns a path that encounters the least objects en route [17]. As an indoor path is essentially a series of indoor partitions (basic topological units like rooms), `FPQ`'s routing cost is partition-passing time, whereas an `LCPQ`'s is partition-passing contact [17].

The fastest path query and the least crowded path query are defined as follows, which are reproduced from [17].

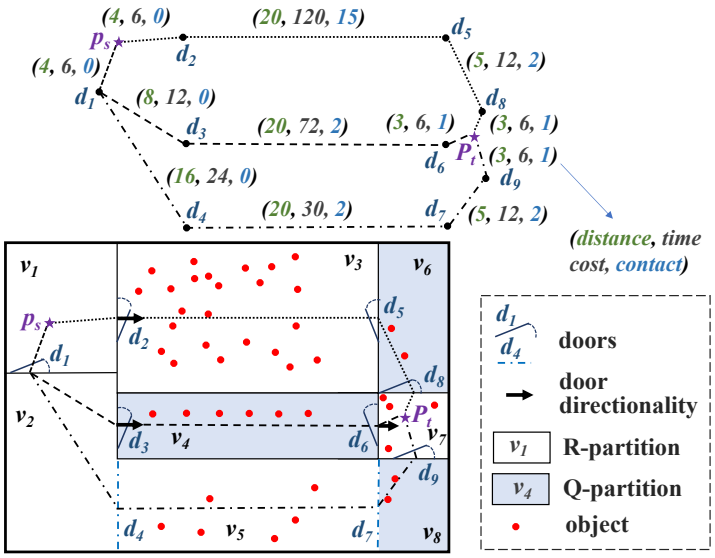


Fig. 5.1: An Example of Floorplan at Query Time t_q

Problem 1 (Indoor Crowd-Aware Fastest Path Query FPQ [17]). *Given a source p_s and a target p_t, an* indoor crowd-aware fastest path query $FPQ(p_s, p_t, t)$ *returns a path $\phi(p_s, d_i, \ldots, d_j, p_t)$ such that a) the overall travel time T_ϕ is minimized and b) ϕ is the shortest among all satisfying a). Formally, $\nexists \phi' \neq \phi$, $T_{\phi'} \leq T_\phi \wedge dist_{\phi'} < dist_\phi$.*

Problem 2 (Indoor Least Crowded Path Query LCPQ [17]). *Given a source p_s and a target p_t, an* indoor least crowded path query $LCPQ(p_s, p_t, t)$ *returns a path $\phi(p_s, d_i, \ldots, d_j, p_t)$ such that a) the overall contact is the least, and b) ϕ is the shortest among all satisfying a). Formally, $\nexists \phi' \neq \phi$, $\kappa_{\phi'} \leq \kappa_\phi \wedge dist_{\phi'} < dist_\phi$.*

5.2 Crowd-Aware Path Planning Framework

We propose a crowd-aware query processing framework as illustrated in Fig. 5.2. It consists of three components, Indoor Crowd Model, Time-evolving Population Estimator, Query Processing Algorithms. We give a detailed introduction of each component in the following sections.

5.2.1 Indoor Crowd Model

Following the accessibility graph [12], we propose the *indoor crowd model* to organize the indoor topology and population [17]. The indoor crowd model $G(V, E, L_V, L_E)$ consists of the following components. 1) The set of vertices V represents indoor partitions. 2) The set of directed edges E where each edge $e(v_i, v_j, d_k) \in E$ means one can

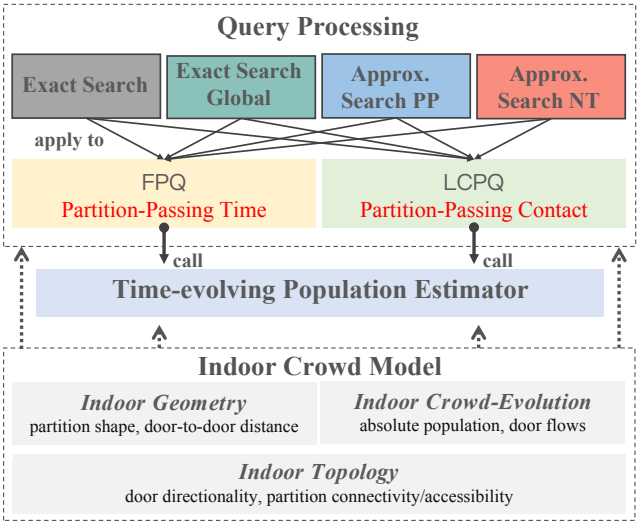


Fig. 5.2: Crowd-Aware Path Planning Framework [17]

reach v_j from v_i through a door d_k. 3) The set of vertex labels L_V where each label is a five tuple $[v_i, Area(v_i), \mathsf{M}_{d2d}, \tau, (\mathsf{P}^i_{t_l}, t_l)]$. Here, v_i represents the corresponding partition, $Area(v_i)$ means this partition's area, M_{d2d} is a matrix maintaining the shortest distance between each pair of doors in this partition. Moreover, $\tau \in \{\mathsf{R}, \mathsf{Q}\}$ represents the type of v_i [1] and $(\mathsf{P}^i_{t_l}, t_l)$ indicates that v_i's absolute population at a latest timestamp t_l is known as $\mathsf{P}^i_{t_l}$. 4) The edge label set where each edge label consists of a **door flow function** $f(v_i, v_j, d_k)$ and a local array $\mathsf{F}[t]$. An example of an indoor crowd model corresponding to the space in Fig. 5.1 is shown in Fig. 5.3.

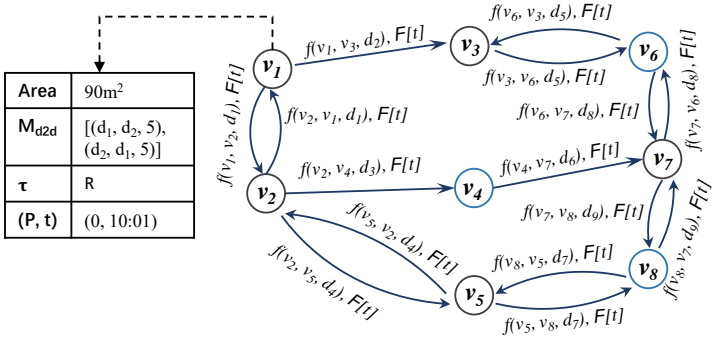


Fig. 5.3: An Example of Indoor Crowd Model [17]

[1] We differentiate two types of partitions. A **Queue Partition** (Q-partition) requests objects to enter and leave *sequentially*, while a **Random Partition** (R-partition) has no such a restriction and objects can enter and leave it *randomly* [17].

We propose a door flow function based on the classic Poisson distribution in queueing theory [6].

$$f(v_i, v_j, d_k) : t \mapsto \mathsf{P}_t, t \in RT(d_k), \mathsf{P}_t \sim Poisson(\lambda) \tag{5.1}$$

In the function, $t \in RT(d_k)$ represents a report timestamp of d_k, P_t denotes the population that flows from v_i to v_j between t and d_k's next report timestamp, and λ represents the expected value of P_t under Poisson distribution [17].

5.2.2 Time-evolving Populations

From the indoor crowd model, a partition v_k's latest population $\mathsf{P}^k_{t_l}$ at an query time t_q earlier time t_l ($t_l \leq t_q$) can be accessed. To estimate the routing cost (traveling time for `FPQ` and contact for `LCPQ`), it is important to derive v_k's time-evolving population and its future inflows/outflows based on $\mathsf{P}^k_{t_l}$ [17].

Let $[t_0, t_1] \in UTI(v_k)$ be the unit time interval covering t_l. We have $\mathsf{P}^k_{t_0,t_1} = \mathsf{P}^k_{t_l}$, meaning that v_k's population over $[t_0, t_1]$ is equal to $\mathsf{P}^k_{t_l}$. Subsequently, for a future unit time interval $[t_x, t_{x+1}] \in UTI(v_k)$, its population can be calculated as

$$\mathsf{P}^k_{t_x,t_{x+1}} = \mathsf{P}^k_{t_{x-1},t_x} - out(v_k, t_x) + in(v_k, t_x),\ x = 1, 2, \ldots \tag{5.2}$$

where $out(v_k, t_x)$ and $in(v_k, t_x)$ are v_k's estimated outflow and inflow at update timestamp t_x, respectively [17].

Assume that all relevant door flow functions are ready at t_q. We can estimate the inflow and outflow at a future update timestamp based on the expected values λ [17]. Formally,

$$out(v_k, t_x) = \sum_{d_i \in P2D_{\sqsubset}(v_k) \wedge t_x \in RT(d_i)} \sum_{v_p \in D2P_{\sqsupset}(d_i)} f(v_k, v_p, d_i).\lambda$$

$$in(v_k, t_x) = \sum_{d_j \in P2D_{\sqsupset}(v_k) \wedge t_x \in RT(d_j)} \sum_{v_q \in D2P_{\sqsubset}(d_j)} f(v_q, v_k, d_j).\lambda$$

where d_i (resp. d_j) is a leaveable (resp. enterable) door updated at time t_x and $v_p \in D2P_{\sqsupset}(d_i)$ (resp. $v_q \in D2P_{\sqsubset}(d_i)$) is its enterable (resp. leaveable) partition [17].

5.3 Query Processing Algorithms

We propose two exact and two approximate algorithms for `FPQ` and `LCPQ` [17].

5.3.1 Exact Algorithms for `FPQ` and `LCPQ`

Two exact query processing algorithms using two different population estimators. The *global estimator* estimates all partitions' populations globally, whereas the *local es-*

timator only estimates a relevant partition's population by looking up its upstream partitions flows [17].

5.3.2 Approximate Algorithms for FPQ and LCPQ

To speed up the query processing, we propose two strategies to derive approximate populations [17]. The following strategies are reproduced from [17].
Strategy 1: Population Derivation for Partial Partitions (PP). When estimating the flows, the estimated flows may be contrary to the real situation, so we need to rectify the expected flows. However, the door flows from a long distance or at a very old timestamp only have a slight impact on a partition's current population. Therefore, we rectify only the outflows of the current relevant partition rather than processing the outflows of its upstream partitions strictly (i.e., the inflows to the current relevant partition) [17].
Strategy 2: Population Derivation at Necessary Timestamps (NT). To further speed up the population derivation for individual partitions, the iterative population computations are skipped. We just estimate its population at the arrival time directly. It is worth noting that Strategy 2 is used in combination with Strategy 1 to achieve the maximum effect of acceleration [17].

5.4 Experimantal Evaluation

We implement four search methods for either FPQ or LCPQ. In particular, *PQ and *PQ-G are exact search algorithms using the local estimator and global estimator, respectively. *PQ-PP and is *PQ-NT are approximate search algorithms using Strategy PP and Strategy NT, respectively. All algorithms are implemented in Java and run on a PC with a 2.30GHz Intel i5 CPU and 16 GB memory [17].

5.4.1 Datasets and Settings

Synthetic Data. We generate a multi-floor indoor space with 141 partitions and 216 doors on each floor [17]. To simulate different size of indoor spaces, we duplicate the floorplan 3, **5**, 7, or 9 times (see Table 5.1). To simulate different types of partitions, we randomly pick 14 partitions having two doors as the Q-partitions while regarding all others as R-partitions on each floor.

Real Data. We use a real dataset that is collected from a seven-floor shopping mall in Hangzhou, China. There are 977 partitions in total, connected by 1613 doors[2]. The flows information is collected from the trajectory records on 2017/01/05. In total, there are 1,598 object trajectories with more than 90,000 positioning records [17].

[2]We assume that there is no Q-partition in this shopping mall. We varied the fraction of Q-partitions/R-partitions on synthetic data, but it shows little impact on all algorithms.

Baseline Methods. We compare our proposals to two baseline methods. One method uses a general time-dependent graph (GTG), where each vertex is a door and the weight of each edge is the cost between two doors, i.e., the time cost for FPQ or the contact for LCPQ [17]. We use a Dijkstra-based algorithm (*PQ-GTG) without pre-computation to keep the fair. To adapt it to our problem, we combine it with our exact population estimator to process FPQ and LCPQ. Another method is the adaptive method based on the indoor crowd model (*PQ-A) [17]. It keeps updating and recomputing the optimal route at every node until the destination is reached [17].

Parameter Settings. Table 5.1 lists all parameter settings with default values in bold[3]. For synthetic data, we simulate the flows and populations. The partition's maximum object number is varied from 0 to $|o|$. The max capacity of a partition v is set as $Area(v) \cdot \beta$ (β is 1 per m^2 in this work). We set the parameter λ of each door flow function as a value ranging from 0 to 3 randomly[4]. Time interval *TI* to control the length of the unit update time interval of partitions [17].

Performance Metrics. To be fair, we run each query instance ten times and compare their average running time and memory cost [17]. To evaluate the accuracy of all methods, we measure the *hit rate* and the *relative error* [17]. The query *hit rate* is the proportion of query instances whose search result equals its gold standard result among all 100 instances[5] [17]. The *relative error* is to measure the accuracy of estimated routing cost against the gold result. The estimated cost refers to overall travel time T_ϕ for FPQ and overall contact κ_ϕ for LCPQ [17]. Taking FPQ as an example, the relative error is $\gamma = |T_\phi^{(E)} - T_\phi^{(G)}| / T_\phi^{(G)}$ where $T_\phi^{(E)}$ and $T_\phi^{(G)}$ is the overall travel time corresponding to the exact search and gold result, respectively [17].

Table 5.1: Parameter Settings [17]

Parameters	Description	Settings
floor	Floor number	3, **5**, 7, 9
$\|o\|$	Partition's maximum object number	300, **600**, 900, 1200, 1500
TI (s)	Time interval	5, **10**, 15, 20
s2t (m)	The shortest distance from p_s to p_t	900, 1100, **1300**, 1500, 1700

5.4.2 Performance Analysis

Comparison in default setting. We report the comparison of algorithms for FPQ and LCPQ on synthetic data in default setting in Tables 5.2 and 5.3. The results in default setting on real data are reported in Tables 5.4 and 5.5. More detailed results report can be seen in Paper D [17]. We summarize the results as follows.

First, in terms of running time and memory use, *PQ-PP and *PQ-NT incur less time and memory cost compared to *PQ and *PQ-G as workloads reduce. Moreover,

[3]We just vary *s2t* for real data.

[4]The value is set according to our analysis of real data. The door flow of a hallway/staircase is relatively more than that of a room.

[5]The gold result is returned by searching over the detailed simulated trajectories.

Table 5.2: Comparison of Algorithms for FPQ on Synthetic Data (best result in bold) [17]

	FPQ	FPQ-G	FPQ-PP	FPQ-NT	FPQ-GTG	FPQ-A
Running Time (ms)	584	585	208	**25**	2857	189
Memory (KB)	115	112	111	**12**	278	14
Hit Rate (%)	**98**	**98**	**98**	95	**98**	94
Relative Error	**4.37E-08**	**4.37E-08**	**4.37E-08**	8.09E-08	**4.37E-08**	0.1233

Table 5.3: Comparison of Algorithms for LCPQ on Synthetic Data (best result in bold) [17]

	LCPQ	LCPQ-G	LCPQ-PP	LCPQ-NT	LCPQ-GTG	LCPQ-A
Running Time (ms)	446	461	131	**20**	2532	163
Memory (KB)	182	192	144	**7**	257	8
Hit Rate (%)	83	83	83	60	83	**87**
Relative Error	**0.0128**	**0.0128**	0.0129	0.1113	**0.0128**	0.1256

Table 5.4: Comparison of Algorithms for FPQ on Real Data (best result in bold) [17]

	FPQ	FPQ-G	FPQ-PP	FPQ-NT	FPQ-GTG	FPQ-A
Running Time (ms)	1900	1997	67	**11**	25559	53
Memory (KB)	367	393	61	**1**	669	2
Hit Rate (%)	**99**	**99**	**99**	98	**99**	98
Relative Error	**1.86E-15**	**1.86E-15**	**1.86E-15**	4.38E-14	**1.86E-15**	0.1492

Table 5.5: Comparison of Algorithms for LCPQ on Real Data (best result in bold) [17]

	LCPQ	LCPQ-G	LCPQ-PP	LCPQ-NT	LCPQ-GTG	LCPQ-A
Running Time (ms)	992	1047	28	**10**	13895	45
Memory (KB)	307	341	30	**1**	568	2
Hit Rate (%)	88	88	88	67	88	**90**
Relative Error	**0.0546**	**0.0546**	**0.0546**	0.6606	**0.0546**	0.062

Strategy NT helps save more time and memory compared to Strategy PP because NT further utilizes historical information to skip timestamps. In terms of hit rate and relative error, PP performs better than NT. That is because NT skips many timestamps, which further decreases the accuracy of intermediate results [17].

Second, FPQ-PP and FPQ-NT outperform LCPQ-PP and LCPQ-NT on accuracy because the partition-passing time is less sensitive to the populations compared to the partition-passing contact [17].

Third, the results show two baseline methods' weaknesses. In particular, *PQ-GTG performs poorly in terms of running time and memory since the graph contains more nodes to process. It seems that *PQ-A performs well on both efficiency and effectiveness. However, it is impossible for a user to obtain the completed path before departure because *PQ-A needs to keep updating during expansion [17].

More experimental results can be seen in an extended version [18].

Chapter 6

Conclusion and Future Work

6.1 Conclusion

This thesis focuses on the spatial queries for indoor location-based services. It summaries four research works. One is an experimental study on indoor spatial queries, while the other three study three novel advanced indoor spatial queries. The contributions of each work are summarized as follows.

- **Paper A** [16] compares five indoor space models or indexes that support four typical indoor spatial queries and give a comprehensive report on experimental study. By analyzing the results, we summarize the pros and cons of all techniques and give recommendations for different indoor typical scenarios.
- **Paper B** [8] studies a new spatial query called Indoor top-k Keyword-aware Routing Query (IKRQ) which can find k routes that have optimal ranking scores integrating keyword relevance and spatial distance constraint. To solve the problem, several techniques are proposed. We propose a keyword mapping method to organize different types of words and manage the relationships between spatial information and keywords. We design two IKRQ search algorithms that expand differently in routing. Extensive experiments are conducted and the results demonstrate the efficiency of our proposals.
- **Paper C** [15] studies a novel routing query called Indoor Temporal-variation aware Shortest Path Query (ITSPQ) which returns the valid shortest path from given points p_s to p_t. We propose a graph (IT-GRAPH) and an index (IT-INDEX) to process the query efficiently. The extensive experiments demonstrate that our IT-INDEX based method performs best because it captures the main shortest distance information in the corresponding tree nodes.
- **Paper D** [17] studies two types of crowd-aware indoor path planning queries. The Indoor Crowd-Aware Fastest Path Query (FPQ) returns a path with the short-

est traveling time in the presence of crowds; the Indoor Least Crowded Path Query (LCPQ) returns a path encountering the least objects en route. To answer these two queries, we design a unified framework including an indoor crowd model, a time-evolving population estimator, and two exact and two approximate query processing algorithms that each can process both query types. The results of the experiments demonstrate the efficiency and scalability of the proposals.

6.2 Future Work

For future work, several interesting directions could be considered.

- **Extending the existing works.** It is interesting to extend the existing work. For example, for Paper C, we may consider designing an index with the automatic update feature and making it more flexible. For Papers B and C, different indoor spatial queries can be considered based on the existing foundations. For all of these works, we could further improve the models or indexes to support other practical issues like elevators/staircases.
- **Using AI techniques.** AI techniques like machine learning models and deep learning models drive related innovation in multiple research areas. These techniques can also be used in indoor spatial queries. For example, it can help predict the indoor crowd or analyze the indoor trajectory, which will be further used in spatial queries.
- **Designing a unified visualization system for indoor space.** Visualization system could provide people with a visual image of indoor space. It is also a tool for knowledge understanding and discovery, which has cognitive analysis and communication functions, and can speed up data processing, e.g., flows in indoor space. It is interesting and significant to design a powerful tool for discovering and understanding scientific laws.

References

[1] https://github.com/indoorLBS/ISQEA.

[2] https://en.wikipedia.org/wiki/Walking.

[3] A. Basiri, E. S. Lohan, T. Moore, A. Winstanley, P. Peltola, C. Hill, P. Amirian, and P. F. e Silva, "Indoor location based services challenges, requirements and usability of current solutions," *Computer Science Review*, vol. 24, pp. 1–12, 2017.

[4] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, "The r*-tree: An efficient and robust access method for points and rectangles," in *Proceedings of the 1990 ACM SIGMOD international conference on Management of data*, 1990, pp. 322–331.

[5] M. A. Cheema, "Indoor location-based services: challenges and opportunities," *SIGSPATIAL Special*, vol. 10, no. 2, pp. 10–17, 2018.

[6] P. C. Consul and G. C. Jain, "A generalization of the poisson distribution," *Technometrics*, vol. 15, no. 4, pp. 791–799, 1973.

[7] C. Costa, X. Ge, and P. Chrysanthis, "Caprio: Context-aware path recommendation exploiting indoor and outdoor information," in *2019 20th IEEE International Conference on Mobile Data Management (MDM)*. IEEE, 2019, pp. 431–436.

[8] Z. Feng, T. Liu, H. Li, H. Lu, L. Shou, and J. Xu, "Indoor top-k keyword-aware routing query," in *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 2020, pp. 1213–1224.

[9] M. Goetz and A. Zipf, "Formal definition of a user-adaptive and length-optimal routing graph for complex indoor environments," *Geo-Spatial Information Science*, vol. 14, no. 2, pp. 119–128, 2011.

[10] S. Huang, J. Cheng, and H. Wu, "Temporal graph traversals: Definitions, algorithms, and applications," *arXiv preprint arXiv:1401.1919*, 2014.

[11] W. Huo and V. J. Tsotras, "Efficient temporal shortest path queries on evolving social graphs," in *Proceedings of the 26th International Conference on Scientific and Statistical Database Management*, 2014, pp. 1–4.

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

[13] H. Li, H. Lu, L. Shou, G. Chen, and K. Chen, "In search of indoor dense regions: An approach using indoor positioning data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 8, pp. 1481–1495, 2018.

[14] T. Liu, Z. Feng, H. Li, H. Lu, M. A. Cheema, H. Cheng, and J. Xu, "Shortest path queries for indoor venues with temporal variations," in *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 2020, pp. 2014–2017.

[15] ——, "Towards indoor temporal-variation aware shortest path query," *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[16] T. Liu, H. Li, H. Lu, M. A. Cheema, and L. Shou, "Indoor spatial queries: Modeling, indexing, and processing," in *Proceedings of the 24th International Conference on Extending Database Technology, EDBT 2021, Nicosia, Cyprus, March 23 - 26, 2021*, 2021, pp. 181–192.

[17] ——, "Towards crowd-aware indoor path planning." *Proc. VLDB Endow.*, vol. 14, no. 8, pp. 1365–1377, 2021.

[18] ——, "Towards crowd-aware indoor path planning (extended version)," *arXiv preprint arXiv:2104.05480*, 2021.

[19] H. Lu, X. Cao, and C. S. Jensen, "A foundation for efficient indoor distance-aware query processing," in *2012 IEEE 28th International Conference on Data Engineering*. IEEE, 2012, pp. 438–449.

[20] H. Lu, C. Guo, B. Yang, and C. S. Jensen, "Finding frequently visited indoor pois using symbolic indoor tracking data," in *19th International Conference on Extending Database Technology*. OpenProceedings. org, 2016, pp. 449–460.

[21] K. Semertzidis and E. Pitoura, "Time traveling in graphs using a graph database." in *EDBT/ICDT Workshops*, 2016, p. 96.

[22] K. Semertzidis, E. Pitoura, and K. Lillis, "Timereach: Historical reachability queries on evolving graphs." in *EDBT*, vol. 15, 2015, pp. 121–132.

[23] Z. Shao, M. A. Cheema, and D. Taniar, "Trip planning queries in indoor venues," *The Computer Journal*, vol. 61, no. 3, pp. 409–426, 2018.

[24] Z. Shao, M. A. Cheema, D. Taniar, and H. Lu, "Vip-tree: an effective index for indoor spatial queries," *Proceedings of the VLDB Endowment*, vol. 10, no. 4, pp. 325–336, 2016.

[25] X. Xie, H. Lu, and T. B. Pedersen, "Efficient distance-aware query evaluation on indoor moving objects," in *2013 IEEE 29th International Conference on Data Engineering (ICDE)*. IEEE, 2013, pp. 434–445.

Part II

Papers

Paper A

Indoor Spatial Queries: Modeling, Indexing, and Processing

Tiantian Liu, Huan Li, Hua Lu, Muhammad Aamir Cheema, Lidan Shou

The paper has been published in
24th International Conference on Extending Database Technology (EDBT),
pp. 181–192, 2021.

Reprinted, with permission, from Tiantian Liu, Huan Li, Hua Lu, Muhammad Aamir Cheema, and Lidan Shou, "Indoor spatial queries: Modeling, indexing, and processing," in 24th International Conference on Extending Database Technology (EDBT), 2021, pp. 181–192.
The layout has been revised.

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.

A.1 Introduction

Indoor location-based services (LBS) are becoming increasingly popular [1, 2]. Relevant applications, such as POI search [3, 4] and routing [5–7], 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 [8–11] 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 [12]. 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 A.2 introduces indoor spatial queries and related work. Sections A.3 and A.4 present the indoor space model/indexes and query processing, respectively. Section A.5 details the experimentation benchmark. Section A.6 reports and analyzes the evaluation results. Section A.7 concludes the paper.

A.2 Indoor Spatial Queries

Table A.1 lists the frequently used notations.

Table A.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

A.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 Fig. A.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 Fig. A.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 Fig. A.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 $p \rightarrow d_{11} \rightarrow o_1$.

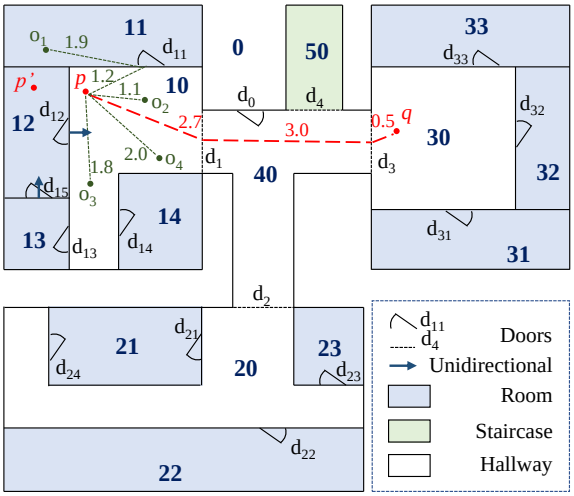


Fig. A.1: Example Floorplan

Lu et al. [8] 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 A.2.1 (Example of Indoor Space Concepts)
In Fig. A.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 $P2D(v_{12}) = \{d_{15}, d_{12}\}$.

A.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 *$RQ(p,r)$ returns all indoor objects from O whose indoor distance from p is within r. Formally, $RQ(p,r) = \{o \mid |p,o|_I \leq r, o \in O\}$.*

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 *$kNNQ(p)$ 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 Fig. A.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 *$SPQ(p,q)$ 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 *$SDQ(p,q)$ returns the shortest indoor distance from p to q, i.e., the length of $SPQ(p,q)$.*

As indicated by the red dashed polyline in Fig. A.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.

A.2.3 Related Work

Indoor Space Modeling. Many indoor space models [13–17] 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. [18] 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. [20] design an indoor moving object index to speed up complex semantic queries in multi-floor spaces. In

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

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

the context of RFID indoor tracking, Yang et al. study continuous range monitoring queries [21] and probabilistic k nearest neighbor queries [22]. To improve the query result, Yu et al. [23] 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. [9, 10] process kNN query and range query for indoor moving objects. Considering uncertain object movements between observed time and query time, Li et al. [4] 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. [16] study historical trajectories of RFID-tracked indoor objects. Delafontaine et al. [24] find sequential visiting patterns within historical Bluetooth tracking data. Given a past time or a time interval, Lu et al. define spatio-temporal joins [25] to find moving object pairs in the same indoor partition, and top-k queries [3] to find the most frequently visited indoor POIs. Ahmed et al. [26, 27] define threshold density query to find dense indoor semantic locations in a historical time interval. Assuming probabilistic sample based location records, Li et al. [28] find the top-k most popular indoor semantic regions with the highest object flow values. Jin et al. [29] study the similarity search over indoor trajectories, considering both spatial and semantic properties. By analyzing spatial constraints of indoor POIs, Jiang et al. [30] study the restoration of indoor trajectories. Li et al. [31] 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 [5] study user-adaptive length-optimal indoor routing based on a weighted routing graph. Salgado et al. [32] study indoor keyword-aware skyline route query, considering the number of covered keywords and route distances. Feng et al. [7] study indoor keyword-aware routing queries to find shortest paths covering user-specified semantic keywords. Costa et al. [6] 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.

A.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.

A.3.1 Indoor Distance-Aware Model

Indoor distance-aware model [8] (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 ∞.

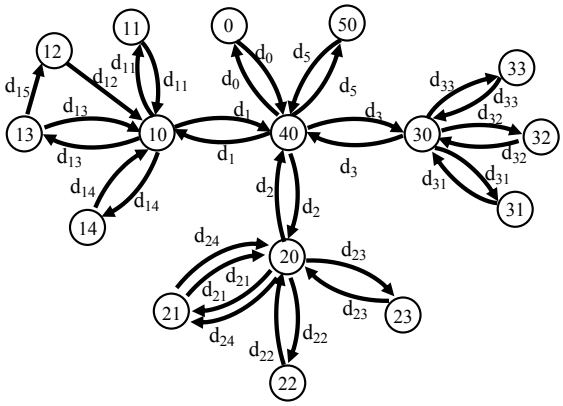
$$f_{\text{d2d}}(v_j, d_i, d_j) = \begin{cases} ||d_i, d_j||_{v_j}, & \text{if } d_i \in P2D_{\sqsupset}(v_j) \\ & \text{and } d_j \in P2D_{\sqsubset}(v_j); \\ 0, & \text{if } d_i = d_j \\ & \text{and } d_i, d_j \in P2D(v_j); \\ \infty, & \text{otherwise.} \end{cases}$$

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.

Fig. A.2 illustrates the IDMODEL for the example shown in Fig. A.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.

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 [8] 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



Door-to-partition distance map

Key	Value
(d_0, v_{40})	*3.5m*
(d_0, v_{30})	*6.1m*
…	

Door-to-door distance map

Key	Value
(v_{40}, d_0, d_1)	*1m*
(v_{40}, d_0, d_5)	*1.2m*
…	

Fig. A.2: An Example of IDMODEL

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.

A.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 [8] (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 $\mathsf{N} = |D|$ 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 Fig. A.1 is illustrated in Fig. A.3. Here, we have $M_{\text{d2d}}[d_1, d_{15}] = 4.6$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}.

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.

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

(a) Distance Matrix M_{d2d}

$$\begin{pmatrix} & 1 & 2 & 3 & 4 & 5 & 6 \\ d_1 & d_1 & d_{11} & d_{12} & d_{14} & d_{13} & d_{15} \\ d_{11} & d_{11} & d_1 & d_{12} & d_{14} & d_{13} & d_{15} \\ d_{12} & d_{12} & d_{15} & d_{14} & d_{11} & d_{13} & d_1 \\ d_{13} & d_{13} & d_{15} & d_{12} & d_{14} & d_1 & d_{11} \\ d_{14} & d_{14} & d_{13} & d_{12} & d_{15} & d_1 & d_{11} \\ d_{15} & d_{15} & d_{13} & d_{12} & d_{14} & d_{11} & d_1 \end{pmatrix}$$

(b) Distance Index Matrix M_{idx}

Fig. A.3: An Example of IDINDEX

A.3.3 Composite Indoor Index

Composite indoor index [9] (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 Fig. A.1 is given in Fig. A.4.

The **geometric layer** uses an R*-tree [35] 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 [9]. Referring to the bottom-right of Fig. A.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 Fig. A.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}.

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 Fig. A.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 [9, 10], the object buckets store static objects in this study. As shown in the bottom-left of Fig. A.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 topolog-

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

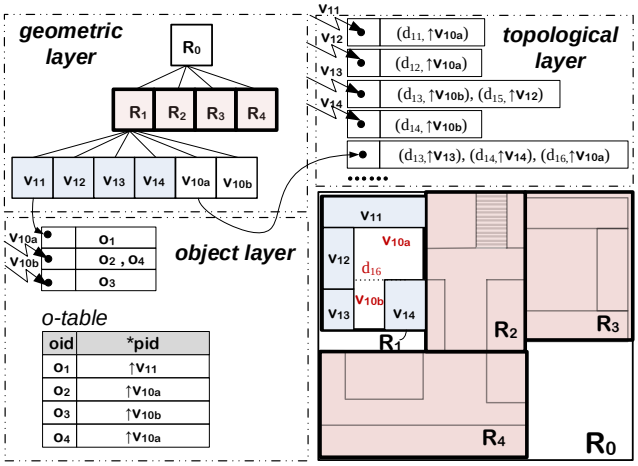


Fig. A.4: CINDEX Example (Adapted from [9])

ical 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.

A.3.4 IP-Tree and VIP-Tree

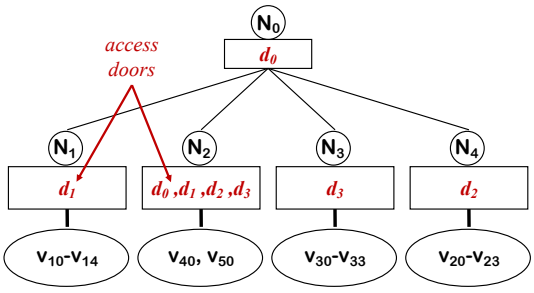
Indoor partitioning tree [11] (IP-TREE) is a tree-based indoor partition index with a number of matrices each materializing the door-to-door distances within a local range. In particular, each leaf node of IP-TREE covers a number of topologically adjacent indoor partitions. The adjacent leaf nodes are 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. $\mathrm{AD}(N)$ 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.

Fig. A.5 shows an example of IP-TREE corresponding to Fig. A.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 $\mathrm{AD}(N_1)$ and $\mathrm{AD}(N_2)$. 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 $\mathrm{d2d}(d_i, d_j)$ and $\mathrm{d2d}(d_j, d_i)$ are kept in each node.

As a variant of IP-TREE, vivid IP-Tree (VIP-TREE) [11] 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)

Fig. A.5: An Example of IP-TREE

A.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 $\mathsf{RQ}(p,r)$ ($k\mathsf{NNQ}(p)$, $\mathsf{SPQ}(p,q)$, or $\mathsf{SDQ}(p,q)$). 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 $\mathsf{RQ}(p,r)$, or kNNs have been found for a $k\mathsf{NNQ}(p)$, or the target point

Table A.2: Feature Comparison

Models	IDMODEL	IDINDEX	CINDEX	IP-TREE	VIP-TREE
Precompute	No	Yes	No	Yes	Yes
Structure	Graph+Mappings	Matrix	Tree+Links	Tree+Matrix	Tree+Matrix
Initialization	Sequential scan	Sequential scan	R*-Tree pruning	Sequential scan	Sequential scan
Expansion	Dijkstra	Loop	Dijkstra	LCA	LCA
RQ	△	✓	✓	✓	✓
*k*NNQ	△	✓	✓	✓	✓
SPQ	✓	△	△	✓	✓
SDQ	✓	△	△	✓	✓

q is met for a $\mathsf{SPQ}(p,q)$ or $\mathsf{SDQ}(p,q)$. Algorithms based on different model/indexes differ in their initializations and expansions. Below, we present a comprehensive analytical comparison of all model/indexes.

A.4.1 Algorithmic Comparison

Table A.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 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 [8] does not provide RQ and *k*NNQ algorithms. Therefore, we implement the two algorithms and refer readers to the appendix in [36]. 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 [37]. 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.

A.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 A.3 summarizes the space complexity of all model/indexes and their time complexity for queries.

Space Complexity. IDMODEL $(V, E_a, L, f_{\text{dv}}, f_{\text{d2d}})$'s space complexity is $\mathscr{O}(\mathtt{V}+\mathtt{Vd}+\mathtt{D}+\mathtt{Vd}+\mathtt{Vd}^2) = \mathscr{O}(\mathtt{Vd}^2)$. IDINDEX's space complexity is $\mathscr{O}(2\mathtt{D}^2) = \mathscr{O}(\mathtt{D}^2)$ as it consists of two door matrices. CINDEX's space complexity is $\mathscr{O}(\mathtt{V}+\mathtt{Vd}+\mathtt{O}) = \mathscr{O}(\mathtt{Vd}+\mathtt{O})$ 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 $\mathscr{O}(\rho\mathtt{D})$ and the latter's is $\mathscr{O}((\rho\mathtt{f})^2\mathtt{L})$ 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 $\mathscr{O}(\rho\mathtt{D}\log_{\mathtt{f}}\mathtt{L})$, 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 $\mathscr{O}(\mathtt{V}\log\mathtt{D})$ and iterates on the objects in each visited partition in $\mathscr{O}(\mathtt{o})$. Also based on graph traversal, the search on CINDEX obtains a subgraph in $\mathscr{O}(\mathtt{V}\log\mathtt{D})$ and visits all objects in each partition of the subgraph in $\mathscr{O}(\mathtt{o})$. 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 $\mathscr{O}(\mathtt{od}\log\mathtt{D})$. The searches via IP-TREE and VIP-TREE work similarly. They prune a tree node based on its distance from the query point in $\mathscr{O}(\log_{\mathtt{f}}\mathtt{L}\cdot\rho\cdot c)$, where c is the unit SDQ cost. Then, they qualify each object in the remaining nodes in $\mathscr{O}(\log_{\mathtt{f}}\mathtt{L}\cdot\mathtt{V}/\mathtt{L}\cdot\mathtt{o}\cdot c)$. Given the SDQ complexity $\mathscr{O}(\rho^2\log_{\mathtt{f}}\mathtt{L})$ for IP-TREE and $\mathscr{O}(\rho^2)$ for VIP-TREE (to be detailed below), their RQ and kNNQ complexities are $\mathscr{O}((\rho\log_{\mathtt{f}}\mathtt{L})^2(\mathtt{Vo}/\mathtt{L}+\rho))$ and $\mathscr{O}(\rho^2\log_{\mathtt{f}}\mathtt{L}(\mathtt{Vo}/\mathtt{L}+\rho))$, respectively.

Time Complexity for SDQ and SPQ. For the graph traversal algorithms of IDMODEL and CINDEX, the SDQ complexity is $\mathscr{O}(\mathtt{V}\log\mathtt{D})$ and SPQ complexity is $\mathscr{O}(\mathtt{V}\log\mathtt{D}+\mathtt{w})$ 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

Table A.3: Complexity Analysis

	Space	RQ	kNNQ	SDQ	SPQ
IDModel	$\mathcal{O}(\mathtt{V}+\mathtt{D}+2\mathtt{V}\mathtt{d}+\mathtt{V}\mathtt{d}^2)$	$\mathcal{O}(\mathtt{o}\mathtt{V}\log\mathtt{D})$	$\mathcal{O}(\mathtt{o}\mathtt{V}\log\mathtt{D})$	$\mathcal{O}(\mathtt{V}\log\mathtt{D})$	$\mathcal{O}(\mathtt{V}\log\mathtt{D}+\mathtt{w})$
IDIndex	$\mathcal{O}(2\mathtt{D}^2)$	$\mathcal{O}(\mathtt{o}\mathtt{d}\log\mathtt{D})$	$\mathcal{O}(\mathtt{o}\mathtt{d}\log\mathtt{D})$	$\mathcal{O}(\mathtt{d}^2)$	$\mathcal{O}(\mathtt{d}^2+\mathtt{w})$
CIndex	$\mathcal{O}(\mathtt{V}+\mathtt{V}\mathtt{d}+\mathtt{0})$	$\mathcal{O}(\mathtt{o}\mathtt{V}\log\mathtt{D})$	$\mathcal{O}(\mathtt{o}\mathtt{V}\log\mathtt{D})$	$\mathcal{O}(\mathtt{V}\log\mathtt{D})$	$\mathcal{O}(\mathtt{V}\log\mathtt{D}+\mathtt{w})$
IP-Tree	$\mathcal{O}(\rho^2\mathtt{f}^2\mathtt{L}+\rho\mathtt{D})$	$\mathcal{O}((\rho\log_\mathtt{f}\mathtt{L})^2(\mathtt{Vo}/\mathtt{L}+\rho))$	$\mathcal{O}((\rho\log_\mathtt{f}\mathtt{L})^2(\mathtt{Vo}/\mathtt{L}+\rho))$	$\mathcal{O}(\rho^2\log_\mathtt{f}\mathtt{L})$	$\mathcal{O}((\rho^2+\mathtt{w})\log_\mathtt{f}\mathtt{L})$
VIP-Tree	$\mathcal{O}(\rho^2\mathtt{f}^2\mathtt{L}+\rho\mathtt{D}\log_\mathtt{f}\mathtt{L})$	$\mathcal{O}(\rho^2\log_\mathtt{f}\mathtt{L}(\mathtt{Vo}/\mathtt{L}+\rho))$	$\mathcal{O}(\rho^2\log_\mathtt{f}\mathtt{L}(\mathtt{Vo}/\mathtt{L}+\rho))$	$\mathcal{O}(\rho^2)$	$\mathcal{O}(\rho^2+\mathtt{w})$

a complexity of $\mathcal{O}(d^2)$. The extra cost of SPQ to concatenate shortest path is of $\mathcal{O}(w)$. 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 $\mathcal{O}(\rho^2 \log_f L)$. In contrast, VIP-TREE materializes the distances from a leaf node to each access door in the ancestors. Its SDQ complexity is $\mathcal{O}(\rho^2)$. The additional cost to construct shortest path in SPQ is $\mathcal{O}(w \log_f L)$ for IP-TREE and $\mathcal{O}(w)$ for VIP-TREE.

A.4.3 Extensibility Analysis

Table A.4 summarizes the extensibility of all model/indexes.

Table A.4: Extensibility Analysis

	IDMODEL	IDINDEX	CINDEX	IP/VIP-TREE
Temporal Variation	✓	X	✓	X
Moving Objects	✓	✓	✓	✓
Uncertain Locations	X	X	✓	X
Keywords	✓	✓	✓	✓

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 [38]. 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 [9, 10] is designed for managing indoor moving objects. It supports distance-aware queries like *k*NNQ 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 [9, 10].

Keywords. A spatial keyword query [39] 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 [7], and boolean *k*NN spatial keyword queries have been supported based on VIP-TREE [40].

A.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 [12].

A.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 Fig. A.6. The data statistics are given in Table A.5.

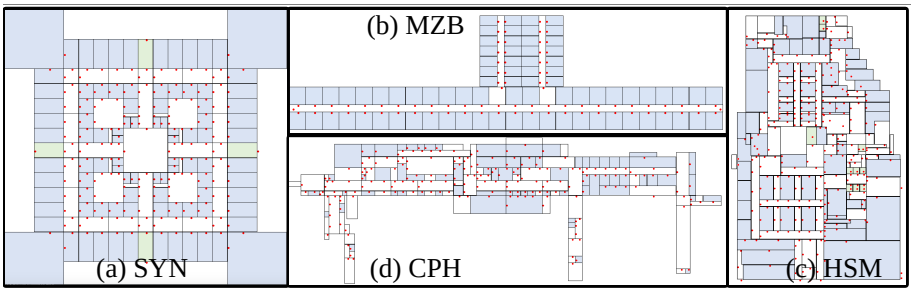


Fig. A.6: Floorplan of Datasets.

Table A.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

Synthetic Building (**SYN**) is a n-floor building. Its each floor is from a real-world floorplan [4] of 1368m $\times$ 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

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

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 $\times$ 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 $\times$ 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.

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

Overall Analysis of Different Datasets. The statistics of the datasets are given in Table A.5. We use $\#dv$ to denote the number of doors in a partition, and conduct quartile statistics [41] on $\#dv$. In Table A.5, $\mathsf{Q1}(\#dv)$, $\mathsf{Q2}(\#dv)$, and $\mathsf{Q3}(\#dv)$ 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 Fig. A.7.

Based on the space scale information and door distribution information from Table A.5 and Fig. A.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 Fig. A.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

[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

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

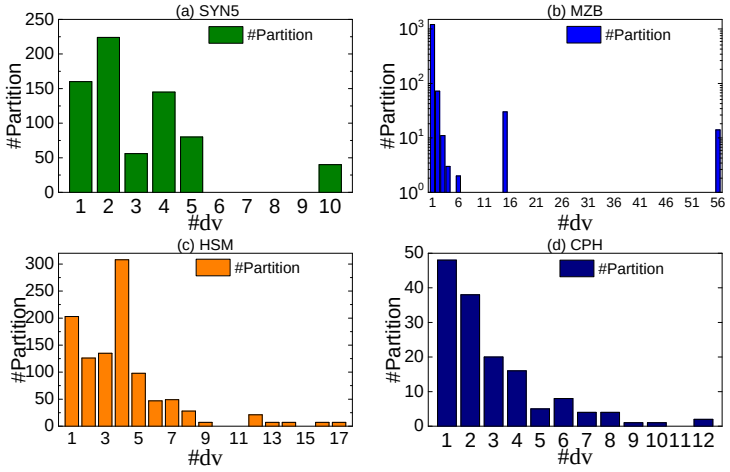


Fig. A.7: Distribution of #*dv* (number of doors in a partition) on (a) SYN5, (b) MZB, (c) HSM, and (d) CPH.

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).

A.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.

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

$k\mathsf{NNQ}(p)$. Similar to RQ generation, we generate ten random kNNQ instances for each k value given in Table A.6.

As SPQ and SDQ can be integrated into one search procedure, we use $\mathsf{SPDQ}(p,q)$ 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.

$\mathsf{SPDQ}(p,q)$. We use a parameter s2t to control the shortest distance from the source p and target q. Its parameter values are listed in Table A.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.

A.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 A.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 [9]. 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 [11]. 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 [11], 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.

A.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 A.6 lists all the evaluation settings. The code of following evaluation procedures and their query instances are also available online [12].

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.

Table A.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$^{\Delta}$

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 *k*NNQ'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^{Δ}.

A.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.

A.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 A.3) and report their size and construction time in Figs. A.8 and A.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 Fig. A.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 (Fig. A.8) and time (Fig. A.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.

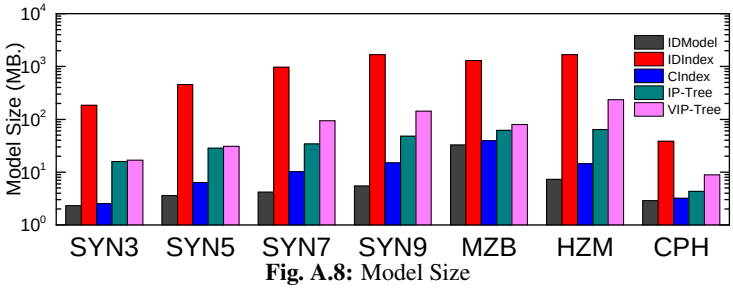


Fig. A.8: Model Size

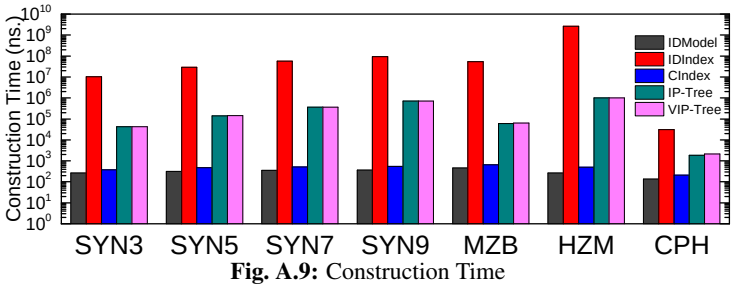


Fig. A.9: Construction Time

A.6.2 Query Processing

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

B1 Effect of Floor Number n (using SYN)

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

- For both query types, IDINDEX always runs fastest as shown in Figs. A.10 and A.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 Figs. A.11 and A.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 A.3, 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 Figs. A.10 and A.12.

- IDMODEL and CINDEX perform similarly, and their execution time increases with a larger n (Figs. A.10 and A.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.

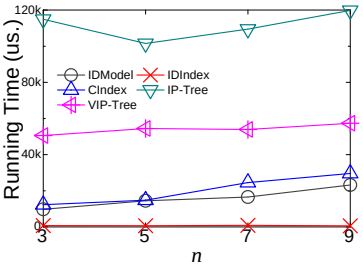


Fig. A.10: RQ Time vs. n

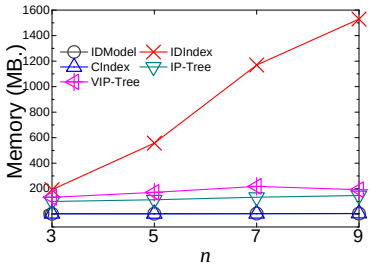


Fig. A.11: RQ Memory vs. n

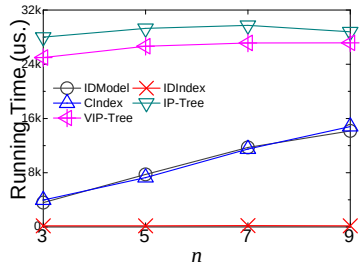


Fig. A.12: kNNQ Time vs. n

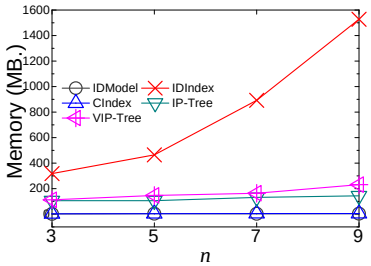


Fig. A.13: kNNQ Memory vs. n

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

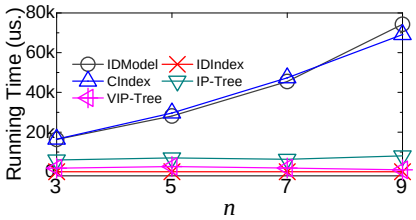


Fig. A.14: SPDQ Time vs. n

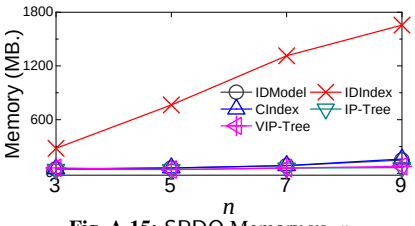


Fig. A.15: SPDQ Memory vs. n

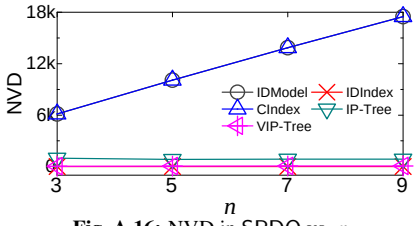


Fig. A.16: NVD in SPDQ vs. n

- 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 (Fig. A.15), and their query performance deteriorates as the space scale increases (Fig. A.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 (Figs. A.14 and A.15).

B2 Effect of Object Number $|O|$

RQ: With different sizes of O, the running time and memory use are reported in Figs. A.17 and A.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 Fig. A.17. This implies that all model/indexes are good at pruning indoor partitions and thus object buckets when processing RQ.
- Referring to Fig. A.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 (Fig. A.18). A special case occurs on CPH (Fig. A.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 Fig. A.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 Fig. A.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 Fig. A.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 Fig. A.18).

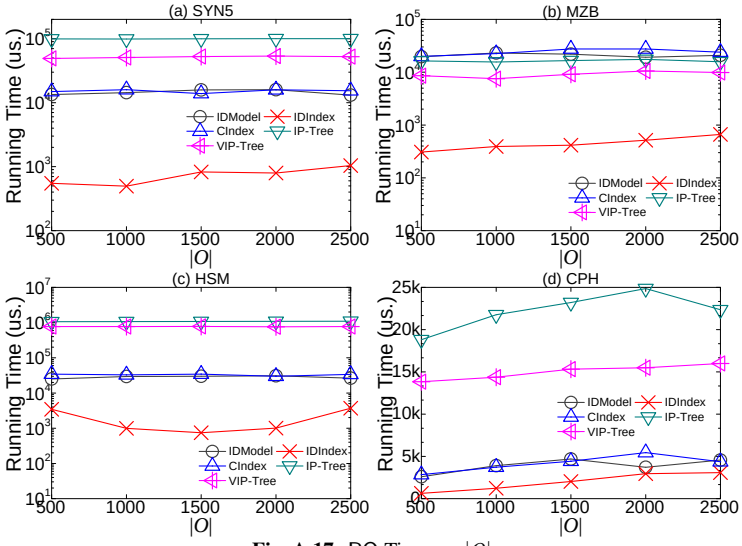


Fig. A.17: RQ Time vs. $|O|$

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

- Referring to Fig. A.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 (Figs. A.19(d) and A.20(d)). However, they run faster on MZB (Fig. A.19(b)) in which many access doors exist due to many C-Pars (see Table A.5).

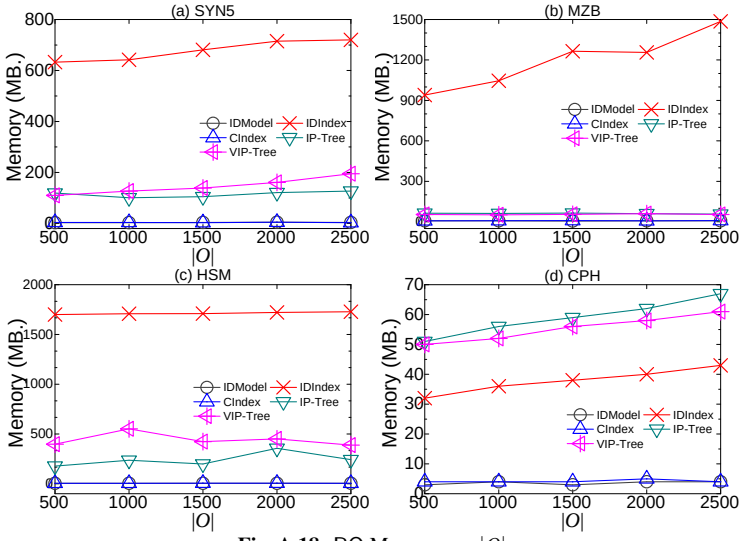


Fig. A.18: RQ Memory vs. $|O|$

- IDMODEL and CINDEX perform comparably as shown in Figs. A.19 and A.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.

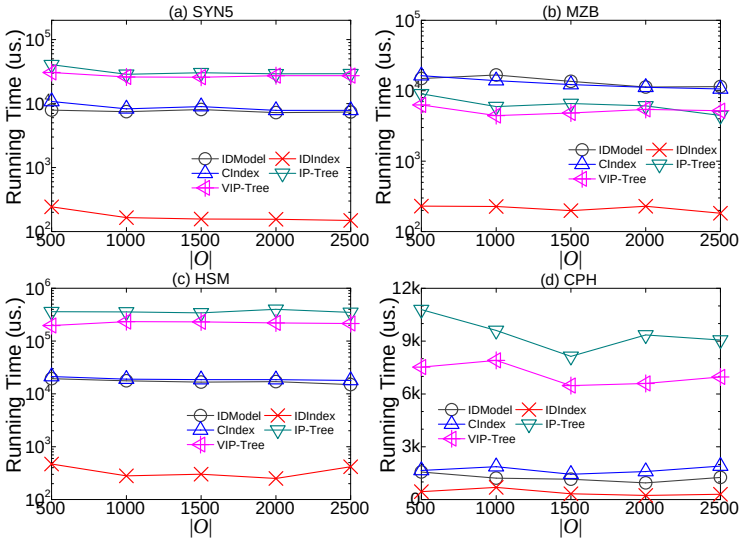


Fig. A.19: kNNQ Time vs. $|O|$

B3 Effect of Range Distance r

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

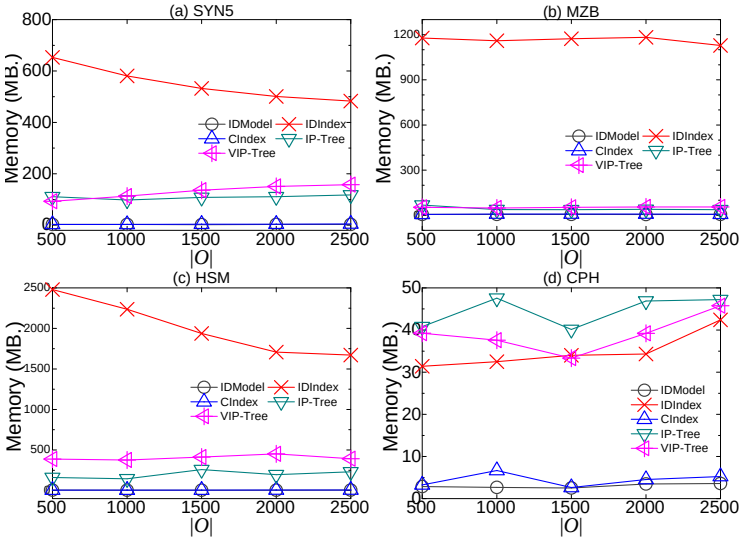


Fig. A.20: kNNQ Memory vs. $|O|$

- On SYN5, MZB, and HSM with complex indoor topology, IDINDEX's running time reported in Fig. A.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 (Fig. A.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 Fig. A.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

kNNQ: The time and memory costs with respect to different k values are reported in Figs. A.23 and A.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 Fig. A.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.

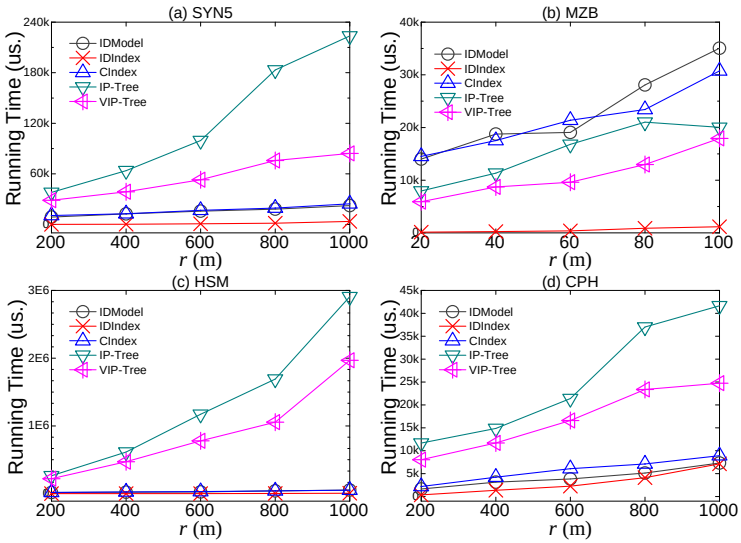


Fig. A.21: RQ Time vs. r

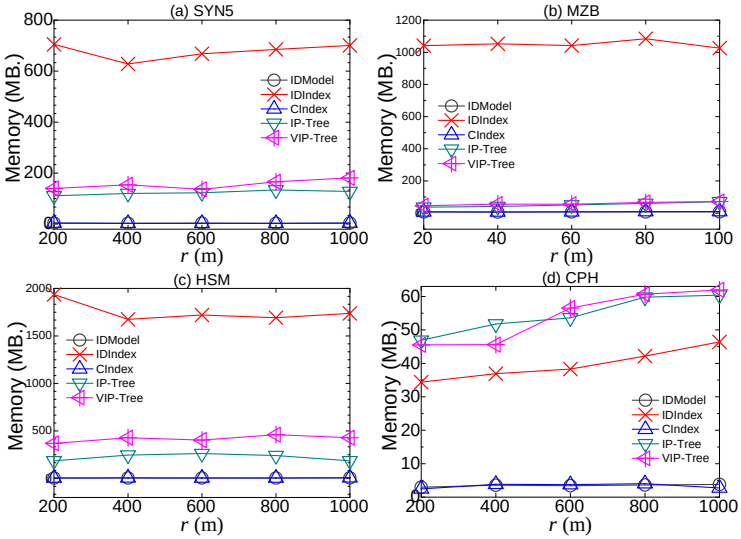


Fig. A.22: RQ Memory vs. r

- Considering both time and memory costs, IDMODEL and CINDEX achieve a good balance when searching for nearest neighbor objects (see Figs. A.23 and A.24).

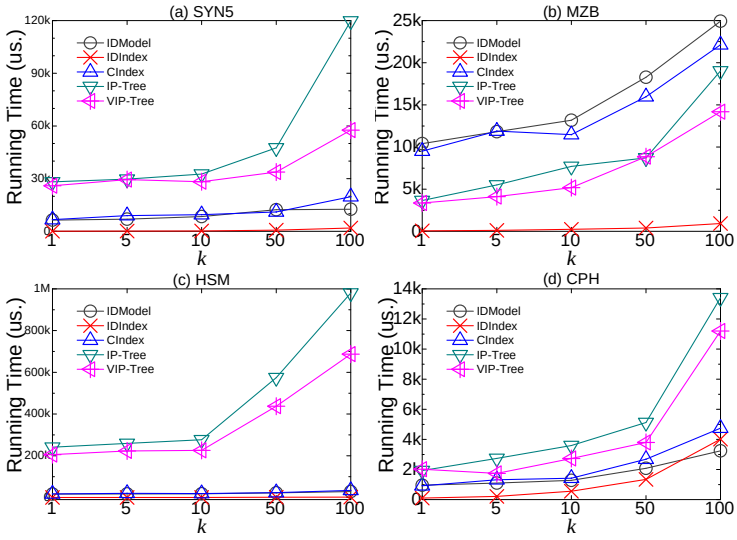


Fig. A.23: kNNQ Time vs. k

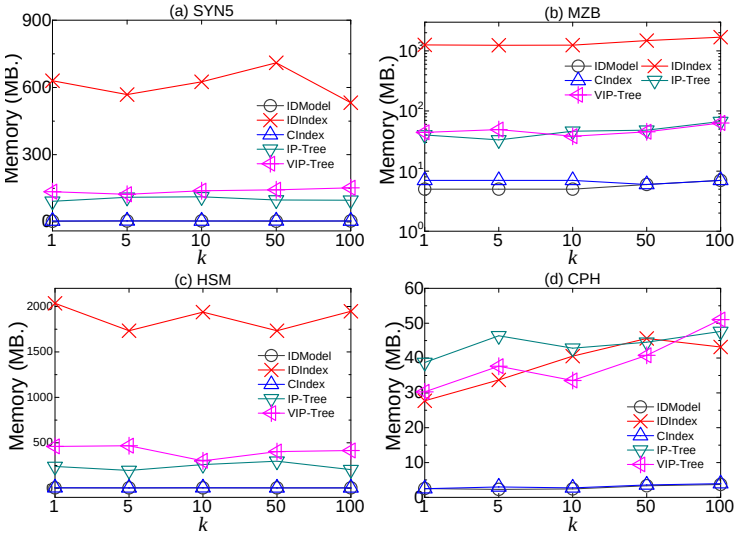


Fig. A.24: kNNQ Memory vs. k

B5 Effect of Source-Target Distance s2t

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

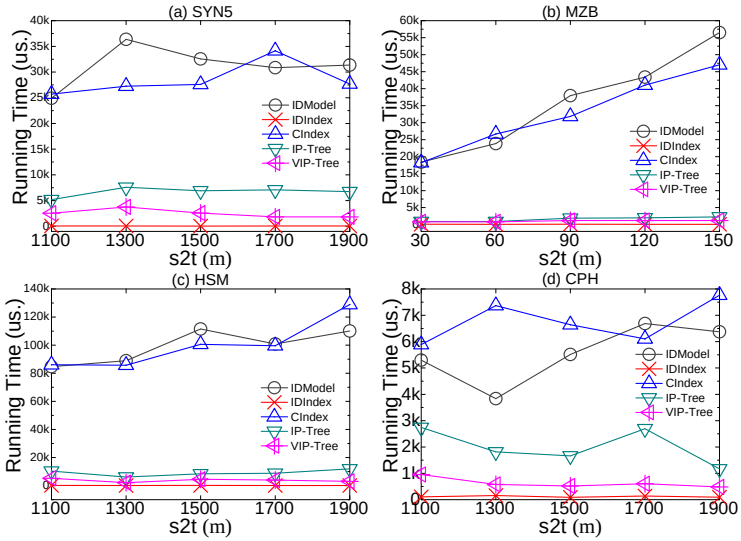


Fig. A.25: SPDQ Time vs. s2t

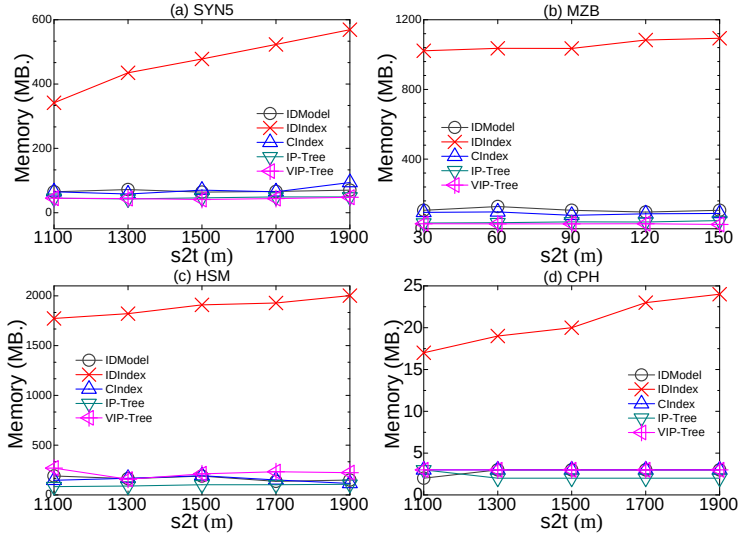


Fig. A.26: SPDQ Memory vs. s2t

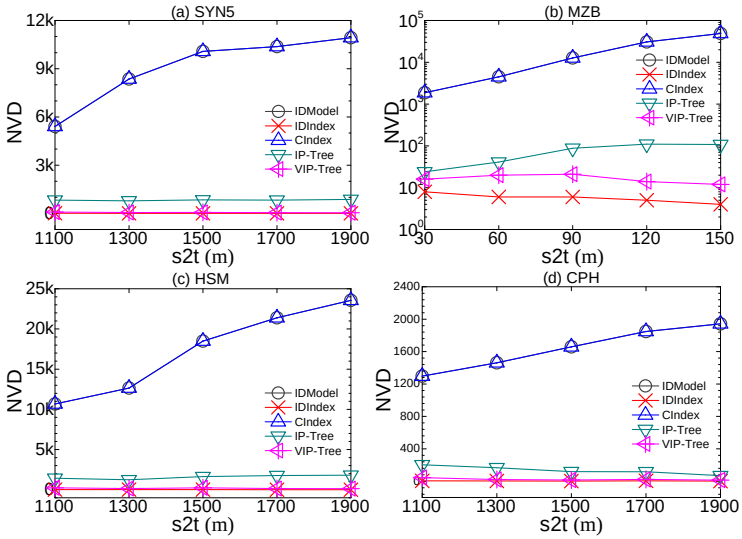


Fig. A.27: NVD in SPDQ vs. s2t

- IDINDEX runs the fastest and is not affected by s2t as reported in Fig. A.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 (Fig. A.27). Nevertheless, its global distance matrix takes up a lot of memory (Fig. A.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 Fig. A.27 are almost the same. Also, as s2t increases, the candidate space becomes larger and the running time of the two becomes longer (see Fig. A.25).

- On MZB and HSM (Fig. A.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.

B6 Effect of Topological Change

RQ and kNNQ: The time cost and memory use with respect to topology characteristics are reported in Tables A.7 and A.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 kNNQ. 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 kNNQ. 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$^+$.

Table A.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 A.8: Results of kNNQ 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

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

- 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.

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 A.10, A.11, A.12, and A.13. For SPDQ, its time cost, memory use and NVD are reported in Tables A.14, A.15, and A.16.

Table A.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

Table A.10: Results (Time) of RQ with Decomposition Method

Model	Time (us.)				
	SYN5^0	**SYN5**	**MZB^0**	**MZB**	**MZB$^\Delta$**
IDMODEL	9695	14999	24065	23527	18917
IDINDEX	460	704	471	439	349
CINDEX	11283	15859	21840	21351	20267
IP-TREE	8923	123076	7957	17215	26110
VIP-TREE	6808	57988	4476	11079	19181

Table A.11: Results (Memory) of RQ with Decomposition Method

Model	Memory (MB.)				
	SYN5^0	**SYN5**	**MZB^0**	**MZB**	**MZB$^\Delta$**
IDMODEL	13	3	12	7	5
IDINDEX	414	437	815	841	1855
CINDEX	12	4	11	8	5
IP-TREE	88	92	61	58	76
VIP-TREE	111	150	62	59	78

Table A.12: Results (Time) of *k*NNQ with Decomposition Method

Model	Time (us.)				
	SYN5^0	**SYN5**	**MZB^0**	**MZB**	**MZB$^\Delta$**
IDMODEL	4773	9240	14318	14224	12828
IDINDEX	143	160	180	185	197
CINDEX	4907	9294	13115	13328	13225
IP-TREE	7272	33693	3904	7315	10369
VIP-TREE	6877	24522	3556	5207	7502

Table A.13: Results (Memory) of *k*NNQ with Decomposition Method

Model	Memory (MB.)				
	SYN5^0	**SYN5**	**MZB^0**	**MZB**	**MZB$^\Delta$**
IDMODEL	9	3	12	6	4
IDINDEX	461	457	679	796	974
CINDEX	16	4	11	8	6
IP-TREE	112	114	36	36	52
VIP-TREE	117	139	43	55	59

- IDINDEX runs fastest when processing RQ and *k*NNQ but uses most memory. When hallways are decomposed into more partitions, IDINDEX's time cost keeps

Table A.14: Results (Time) of SPDQ with Decomposition Method

Model	Time (us.)				
	SYN5^0	**SYN5**	**MZB^0**	**MZB**	**MZB$^\Delta$**
IDMODEL	31242	31855	36220	33503	32396
IDINDEX	138	75	71	69	73
CINDEX	32823	26900	35307	33238	31806
IP-TREE	1610	7523	1252	1893	3257
VIP-TREE	856	2379	1091	1126	1474

Table A.15: Results (Memory) of SPDQ with Decomposition Method

Model	Memory (MB.)				
	SYN5^0	**SYN5**	**MZB^0**	**MZB**	**MZB$^\Delta$**
IDMODEL	44	63	54	73	92
IDINDEX	388	396	1096	1273	1489
CINDEX	43	65	52	97	110
IP-TREE	59	44	31	39	44
VIP-TREE	64	42	54	39	55

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 *k*NNQ. 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 *k*NNQ. Moreover, the time cost of IP-TREE and VIP-TREE rises faster when processing RQ and *k*NNQ than processing SPDQ. That is because there is some extra cost to prune nodes when processing RQ and *k*NNQ. As the nodes increase, this extra cost increases fast.

A.6.3 Summary of Findings

We summarize all five model/indexes' performance in Table A.17 where more stars imply a better performance (lower cost). IDMODEL incurs minimum time and space costs in construction. It works well for RQ and *k*NNQ, 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

Table A.16: Results (NVD) of SPDQ with Decomposition Method

Model	NVD				
	SYN5^0	**SYN5**	**MZB^0**	**MZB**	**MZB$^\Delta$**
IDMODEL	82574	10074	24877	12718	4243
IDINDEX	58	8	22	9	8
CINDEX	82574	10074	24877	12718	4243
IP-TREE	416	843	97	87	139
VIP-TREE	136	61	37	24	26

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 A.17: 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	⋆⋆	⋆⋆⋆	⋆⋆⋆	⋆⋆	⋆⋆⋆	⋆⋆⋆⋆

A.7 Conclusion and Future Work

This work reports on an extensive experimental evaluation of five indoor space model or 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] A. Basiri, E. S. Lohan, T. Moore, A. Winstanley, P. Peltola, C. Hill, P. Amirian, and P. F. e Silva, "Indoor location based services challenges, requirements and usability of current solutions," *Computer Science Review*, vol. 24, pp. 1–12, 2017.

[2] M. A. Cheema, "Indoor location-based services: challenges and opportunities," *SIGSPATIAL Special*, vol. 10, no. 2, pp. 10–17, 2018.

[3] H. Lu, C. Guo, B. Yang, and C. S. Jensen, "Finding frequently visited indoor pois using symbolic indoor tracking data," in *19th International Conference on Extending Database Technology*. OpenProceedings. org, 2016, pp. 449–460.

[4] H. Li, H. Lu, L. Shou, G. Chen, and K. Chen, "In search of indoor dense regions: An approach using indoor positioning data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 8, pp. 1481–1495, 2018.

[5] M. Goetz and A. Zipf, "Formal definition of a user-adaptive and length-optimal routing graph for complex indoor environments," *Geo-Spatial Information Science*, vol. 14, no. 2, pp. 119–128, 2011.

[6] C. Costa, X. Ge, and P. Chrysanthis, "Caprio: Context-aware path recommendation exploiting indoor and outdoor information," in *2019 20th IEEE International Conference on Mobile Data Management (MDM)*. IEEE, 2019, pp. 431–436.

[7] Z. Feng, T. Liu, H. Li, H. Lu, L. Shou, and J. Xu, "Indoor top-k keyword-aware routing query," in *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 2020, pp. 1213–1224.

[8] H. Lu, X. Cao, and C. S. Jensen, "A foundation for efficient indoor distance-aware query processing," in *2012 IEEE 28th International Conference on Data Engineering*. IEEE, 2012, pp. 438–449.

[9] X. Xie, H. Lu, and T. B. Pedersen, "Efficient distance-aware query evaluation on indoor moving objects," in *2013 IEEE 29th International Conference on Data Engineering (ICDE)*. IEEE, 2013, pp. 434–445.

[10] ——, "Distance-aware join for indoor moving objects," *IEEE Transactions on Knowledge and Data Engineering*, vol. 2, no. 27, pp. 428–442, 2015.

[11] Z. Shao, M. A. Cheema, D. Taniar, and H. Lu, "Vip-tree: an effective index for indoor spatial queries," *Proceedings of the VLDB Endowment*, vol. 10, no. 4, pp. 325–336, 2016.

[12] https://github.com/indoorLBS/ISQEA.

[13] J. Lee, "A spatial access-oriented implementation of a 3-d gis topological data model for urban entities," *GeoInformatica*, vol. 8, no. 3, pp. 237–264, 2004.

[14] E. Whiting, J. Battat, and S. Teller, "Topology of urban environments," in *Computer-Aided Architectural Design Futures (CAADFutures) 2007*. Springer, 2007, pp. 114–128.

[15] T. Becker, C. Nagel, and T. H. Kolbe, "A multilayered space-event model for navigation in indoor spaces," in *3D geo-information sciences*. Springer, 2009, pp. 61–77.

[16] C. S. Jensen, H. Lu, and B. Yang, "Indexing the trajectories of moving objects in symbolic indoor space," in *International symposium on spatial and temporal databases*. Springer, 2009, pp. 208–227.

[17] M. Worboys, "Modeling indoor space," in *Proceedings of the 3rd ACM SIGSPATIAL international workshop on indoor spatial awareness*, 2011, pp. 1–6.

[18] S. Alamri, D. Taniar, and M. Safar, "Indexing moving objects in indoor cellular space," in *2012 15th International Conference on Network-Based Information Systems*. IEEE, 2012, pp. 38–44.

[19] Y. Kim, H. Jung, J. Jang, and U.-M. Kim, "An efficient grid index for moving objects in indoor environments," in *Proceedings of the 10th international conference on ubiquitous information management and communication*, 2016, pp. 1–4.

[20] H. Lin, L. Peng, S. Chen, T. Liu, and T. Chi, "Indexing for moving objects in multi-floor indoor spaces that supports complex semantic queries," *ISPRS International Journal of Geo-Information*, vol. 5, no. 10, p. 176, 2016.

[21] B. Yang, H. Lu, and C. S. Jensen, "Scalable continuous range monitoring of moving objects in symbolic indoor space," in *Proceedings of the 18th ACM conference on Information and knowledge management*, 2009, pp. 671–680.

[22] ——, "Probabilistic threshold k nearest neighbor queries over moving objects in symbolic indoor space," in *Proceedings of the 13th international conference on extending database technology*, 2010, pp. 335–346.

[23] J. Yu, W.-S. Ku, M.-T. Sun, and H. Lu, "An rfid and particle filter-based indoor spatial query evaluation system," in *Proceedings of the 16th International Conference on Extending Database Technology*, 2013, pp. 263–274.

[24] M. Delafontaine, M. Versichele, T. Neutens, and N. Van de Weghe, "Analysing spatiotemporal sequences in bluetooth tracking data," *Applied Geography*, vol. 34, pp. 659–668, 2012.

[25] H. Lu, B. Yang, and C. S. Jensen, "Spatio-temporal joins on symbolic indoor tracking data," in *2011 IEEE 27th International Conference on Data Engineering*. IEEE, 2011, pp. 816–827.

[26] T. Ahmed, T. B. Pedersen, and H. Lu, "Finding dense locations in indoor tracking data," in *2014 IEEE 15th International Conference on Mobile Data Management*, vol. 1. IEEE, 2014, pp. 189–194.

[27] ——, "Finding dense locations in symbolic indoor tracking data: modeling, indexing, and processing," *GeoInformatica*, vol. 21, no. 1, pp. 119–150, 2017.

[28] H. Li, H. Lu, L. Shou, G. Chen, and K. Chen, "Finding most popular indoor semantic locations using uncertain mobility data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 11, pp. 2108–2123, 2018.

[29] P. Jin, T. Cui, Q. Wang, and C. S. Jensen, "Effective similarity search on indoor moving-object trajectories," in *International conference on database systems for advanced applications*. Springer, 2016, pp. 181–197.

[30] X. Jiang, Y. Xing, Y. Chen, Y. Gu, and J. Liu, "Indoor trajectory restoration method based on poi relation constraints," in *2019 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*. IEEE, 2019, pp. 439–446.

[31] H. Li, H. Lu, M. A. Cheema, L. Shou, and G. Chen, "Indoor mobility semantics annotation using coupled conditional markov networks," in *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 2020, pp. 1441–1452.

[32] C. Salgado, "Keyword-aware skyline routes search in indoor venues," in *Proceedings of the 9th ACM SIGSPATIAL International Workshop on Indoor Spatial Awareness*, 2018, pp. 25–31.

[33] N. Sun, E. Yang, J. Corney, and Y. Chen, "Semantic path planning for indoor navigation and household tasks," in *Annual Conference Towards Autonomous Robotic Systems*. Springer, 2019, pp. 191–201.

[34] Z. Wang, H. Xie, Z. Lin, T. Wen, C. Guo, and H. Chen, "The robot path planning algorithm in indoor environment," in *IECON 2020 The 46th Annual Conference of the IEEE Industrial Electronics Society*. IEEE, 2020, pp. 5350–5355.

[35] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, "The r*-tree: An efficient and robust access method for points and rectangles," in *Proceedings of the 1990 ACM SIGMOD international conference on Management of data*, 1990, pp. 322–331.

[36] T. Liu, H. Li, H. Lu, M. A. Cheema, and L. Shou, "An experimental analysis of indoor spatial queries: Modeling, indexing, and processing," *arXiv preprint arXiv:2010.03910*, 2020.

[37] D. B. Johnson, "A note on dijkstra's shortest path algorithm," *Journal of the ACM (JACM)*, vol. 20, no. 3, pp. 385–388, 1973.

[38] T. Liu, Z. Feng, H. Li, H. Lu, M. A. Cheema, H. Cheng, and J. Xu, "Shortest path queries for indoor venues with temporal variations," in *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 2020, pp. 2014–2017.

[39] L. Chen, G. Cong, C. S. Jensen, and D. Wu, "Spatial keyword query processing: An experimental evaluation," *Proceedings of the VLDB Endowment*, vol. 6, no. 3, pp. 217–228, 2013.

[40] Z. Shao, M. A. Cheema, D. Taniar, H. Lu, and S. Yang, "Efficiently processing spatial and keyword queries in indoor venues," *IEEE Transactions on Knowledge and Data Engineering*, 2020.

[41] D. G. Altman and J. M. Bland, "Statistics notes: quartiles, quintiles, centiles, and other quantiles," *Bmj*, vol. 309, no. 6960, pp. 996–996, 1994.

Paper B

Indoor Top-k Keyword-aware Routing Query

Zijin Feng, Tiantian Liu, Huan Li, Hua Lu, Lidan Shou, Jianliang Xu

The paper has been published in the
IEEE 36th International Conference on Data Engineering (ICDE), pp. 1213–1224, 2020.

Abstract

People have many activities indoors and there is an increasing demand of keyword-aware route planning for indoor venues. In this paper, we study the indoor top-k keyword-aware routing query (IKRQ). Given two indoor points s and t, an IKRQ returns k s-to-t routes that do not exceed a given distance constraint but have optimal ranking scores integrating keyword relevance and spatial distance. It is challenging to efficiently compute the ranking scores and find the best yet diverse routes in a large indoor space with complex topology. We propose prime routes to diversify top-k routes, devise mapping structures to organize indoor keywords and compute route keyword relevances, and derive pruning rules to reduce search space in routing. With these techniques, we design two search algorithms with different routing expansions. Experiments on synthetic and real data demonstrate the efficiency of our proposals.

B.1 Introduction

Route planning is among the popular location-based services. Recently, it is increasingly in demand in various indoor venues such as shopping malls, railway stations and airports. As such venues accommodate significant parts of people's daily life, appropriate route planning can facilitate a huge number of people, especially when they have to go through a large and/or unfamiliar indoor environment.

Take Copenhagen Airport as an example. Suppose Jesper has just passed the security check for his flight to France. En route to his boarding gate, he wants to buy some Danish cookies, draw some euros in cash, and eat a bowl of noodle. He wants to reach the gate within 1.5 hours. His needs can be represented as an indoor routing query from a start point (security check) to a terminal point (his boarding gate). A desirable route should have a shop that sells cookies, an ATM or a bank that offers euros, and a restaurant offers noodles. The route should not be too long, i.e., the route distance should be less than a distance constraint.[1]

Indoor route planning is also applicable in other practical scenarios. For example, by specifying a request with keywords of "coffee" and "print", a person in an office can have a service robot to fetch a cup of coffee and a printout document in one single route. Moreover, in automatic warehouses of Amazon, JD.com and Alibaba, robots can make use of indoor routing with keywords to accomplish operational tasks, e.g., fetching or delivering particular products at particular locations.

In this paper, we formulate and study indoor top-k keyword-aware routing query (IKRQ). An IKRQ requires a start point s, a terminal point t, a distance constraint Δ, and a query keyword list QW. It returns the k best routes from s to t that are not longer than Δ and have highest ranking scores. A route score integrates the route's keyword relevance w.r.t. QW and its route distance, i.e., length from s to t.

[1] A time constraint T, e.g., 1.5 hours, can easily be converted to a distance constraint $\Delta = V_{max} \cdot T$, where V_{max} is the maximum indoor walking speed.

We differentiate two kinds of indoor keywords. An identity word (i-word) is the semantic name for an indoor partition[2]; a thematic word (t-word) further describes an i-word's partition. In the airport example above, the specific shop names, restaurant names, and ATM are i-words. T-words can be different things for different i-words. A shop's t-words can be the names of its goods. A restaurant's t-words can be the dishes on its menu. An ATM's t-words can be *Danish krone*, *euro* and *Swedish krone* that indicate available currencies in cash. This keyword differentiation makes more sense indoors than outdoors. When inside an indoor venue, people tend to visit a point-of-interest (POI) with a particular name, e.g., *Apple* or *Samsung*. In contrast, outdoor routing cannot benefit from venue names as they carry little semantics; neither can it work with keywords for indoor partitions as the partitions in the same venue are regarded as co-located in outdoor space.

An IKRQ is non-trivial due to several factors. First, it needs to define keyword relevance for routes w.r.t. query keywords, for which we need to consider both i-words and t-words in the indoor context. Second, it needs to integrate keyword relevance and spatial distance for ranking routes. A meaningful ranking score may be expensive to compute for routes with multiple hops. Third, it needs to search for routes in an indoor venue with a large number of partitions that form complex topology, which may result in a large search space for routing.

To resolve IKRQs, we develop a set of techniques. First, the concept of prime routes diversifies top-k routes in the query result, and enables a particular pruning rule to reduce search space. Second, bi-directional mapping structures organize two-level indoor keywords, which facilitates computing keyword relevances and ranking scores for routes that involve indoor partitions. Third, other pruning rules are derived based on the distance constraint and the bound of top-k result. Fourth, two search algorithms are designed for routing, employing a topology-oriented expansion (ToE) and a keyword-oriented expansion (KoE), respectively. All proposed techniques are experimentally evaluated on synthetic and real data. The experimental results demonstrate the efficiency of our proposals and disclose the respective suitable settings for ToE and KoE.

We make the following contributions in this paper:

- We formulate indoor top-k keyword-aware routing query (IKRQ). We also propose prime routes that diversify top-k results and reduce search space. (Section B.2)
- We propose a scheme to organize the indoor keywords, a method to compute keyword relevance for routes, and a ranking score for routes. (Section B.3)
- We derive a set of pruning rules for IKRQ search, and design a unified search framework with two algorithms that expand differently in routing. (Section B.4)
- We conduct extensive experiments on synthetic and real data sets to evaluate our proposals. (Section B.5)

[2]A partition is a basic indoor region with clear boundaries. Examples are rooms, staircases, and booths.

In addition, we review the related work in Section B.6 and conclude the paper in Section B.7.

B.2 Problem Formulation

B.2.1 Preliminaries

Table B.1 lists the frequently used notations.

Table B.1: Notations

Symbol	Meaning
v, d, p	partition, door, and point in an indoor space
w_i, w_t	an identity word, a thematic word
$PW(v_i)$	partition words of partition v_i
QW	query keyword list
$KP(R_i)$	sequence of key partitions on route R_i
$RW(R_i)$	route words of route R_i
$\kappa(w_Q)$	candidate i-word set of query keyword w_Q
$\rho_{QW}(R_i)$	keyword relevance of route R_i w.r.t. QW
$\psi(R_i)$	ranking score of route R_i

A previous work [1] defines mappings that capture indoor topology. 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. As a basic step, indoor routing needs to move from one door to another through their common partition. To this end, we have intra-partition **door-to-door distance** for two doors d_i and d_j as

$$\delta_{d2d}(d_i,d_j) = \begin{cases} |d_i,d_j|_E, & \text{if } D2P_{\sqsupset}(d_i) \cap D2P_{\sqsubset}(d_j) \neq \varnothing; \\ \infty, & \text{otherwise.} \end{cases}$$

Here, $D2P_{\sqsupset}(d_i) \cap D2P_{\sqsubset}(d_j) \neq \varnothing$ means d_i and d_j are in the same partition that one can enter via d_i and leave via d_j. In this case, we measure the Euclidean distance between d_i and d_j. The case of $d_i = d_j$ is special. This happens when one needs to enter a partition due to its keyword relevance but then leave it from the same door for further routing. In this case, we set $\delta_{d2d}(d_i,d_j)$ to be the double of the longest non-loop distance one can reach inside the partition from the pertinent door. Note that δ_{d2d} simplifies the f_{d2d} function [1] such that no partition is explicitly specified.

Moreover, the previous work [1] uses $v(p_i)$ to denote point p_i's host partition, $P2D_{\sqsupset}(v_k)$ the set of *enterable* doors through which one can enter partition v_k, and $P2D_{\sqsubset}(v_k)$ the set of *leaveable* doors through which one can leave partition v_k. Still within a partition, we define **point-to-door distance** and **door-to-point distance**, re-

spectively, as follows. Given a door d_k, two points p_i and p_j, we have

$$\delta_{pt2d}(p_i,d_k) = \begin{cases} |p_i,d_k|_E, & \text{if } d_k \in P2D_{\sqsubset}(v(p_i)); \\ \infty, & \text{otherwise.} \end{cases}$$

$$\delta_{d2pt}(d_k,p_i) = \begin{cases} |d_k,p_j|_E, & \text{if } d_k \in P2D_{\sqsupset}(v(p_j)); \\ \infty, & \text{otherwise.} \end{cases}$$

The two distances also facilitate indoor routing: $\delta_{pt2d}(p_i,d_k)$ is the intra-partition distance from point p_i to door d_k when one leaves the partition; $\delta_{d2pt}(d_k,p_j)$ is the intra-partition distance from door d_k to point p_i when one enters the partition.

We generalize doors and points to *items* represented by x. When the specific item types are unclear or not important, we use $\delta_*(x_i,x_j)$ to indicate one of δ_{d2d}, δ_{d2pt} and δ_{pt2d}.

B.2.2 Principles and Definition of Routing Query

Definition 1 (Route and Route Distance). *A* **route** $R = (x_s,d_i,\ldots,d_n,x_t)$ *is a path through a sequence of doors from an item x_s to an item x_t, where x_s and x_t can be a point or a door. Given routing request, R is a* **complete route** *if x_s and x_t are the start and terminal points, respectively. Otherwise, we call it a* **partial route**. *A route R's* **route distance** *is* $\delta(R) = \delta_*(x_s,d_i) + \sum_{k=i}^{n-1} \delta_*(d_k,d_{k+1}) + \delta_*(d_n,x_t)$.

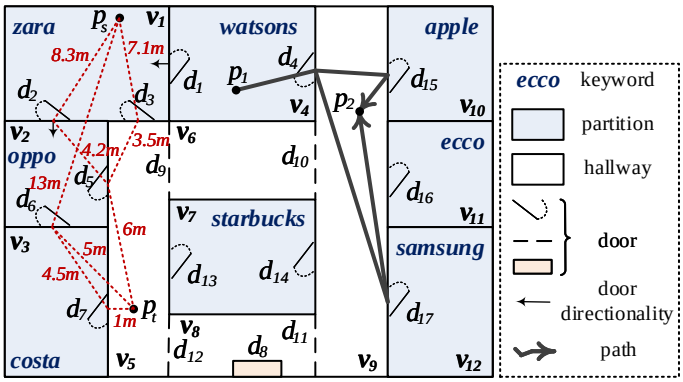


Fig. B.1: An Example of Floorplan

Example B.2.1 (Example of Route and Route Distance)
Referring to Fig. B.1, one can start from p_s in partition v_1, pass doors d_2 and d_5, and reach p_t in partition v_5. The complete route is $R = (p_s,d_2,d_5,p_t)$, and a partial route is $R^\star = (p_s,d_2,d_5)$. Assuming $\delta_{pt2d}(p_s,d_2)$ = 8.3m, $\delta_{d2d}(d_2,d_5)$ = 4.2m, and $\delta_{d2pt}(d_5,p_t)$ = 6m, we have $\delta(R^\star) = \delta_{pt2d}(p_s,d_2) + \delta_{d2d}(d_2,d_5)$ = 12.5m and $\delta(R)$ = 18.5m.

Using the topological mappings introduced in Section B.2.1, we can easily obtain the partitions that a route R passes. For example, given $R^\star = (p_s, d_2, d_5)$ we know that $R^\star$ passes v_1 between (p_s, d_2) and v_2 between (d_2, d_5). Before we formulate our indoor routing query, we discuss two principles of indoor route search.

Principle of Regularity. Traditional outdoor routing algorithms [2–5] usually exclude loops in a route. This regularization avoids endless route searching. However, a regular route in indoor space can have a loop of doors within one-hop. Referring to Fig. B.1, anyone who needs to visit partition v_{10} must enter and then leave d_{15}, the only accessible door of v_{10}. Accordingly, the principle of regularity disqualifies a route that contains one or more doors between two identical doors. For example, partial route $(d_{13}, d_{14}, d_{14}, d_{13})$ is not allowed, because of the doors between the two appearances of door d_{13}. This partial route means that one starts from d_{13}, passes v_7 twice and returns to d_{13} again.

Principle of Diversity. The idea of diversifying top-k results [6] inspires us to avoid homogeneous routes in our indoor routing. Back to the example in Fig. B.1, suppose a user wants routes from p_s to p_t while covering two keywords *oppo* and *costa*. Several possible routes are listed in Table B.2. For ease of reading, we insert between each two consecutive route items the partition that connects the two items.

Table B.2: Examples of Routes from p_s to p_t

R_1	$(p_s \xrightarrow{v_1} d_2 \xrightarrow{v_2} d_6 \xrightarrow{v_3} d_7 \xrightarrow{v_5} p_t)$
R_2	$(p_s \xrightarrow{v_1} d_2 \xrightarrow{v_2} d_5 \xrightarrow{v_5} d_7 \xrightarrow{v_3} d_7 \xrightarrow{v_5} p_t)$
R_3	$(p_s \xrightarrow{v_1} d_2 \xrightarrow{v_2} d_5 \xrightarrow{v_5} d_9 \xrightarrow{v_6} d_9 \xrightarrow{v_5} d_7 \xrightarrow{v_3} d_7 \xrightarrow{v_5} p_t)$
R_4	$(p_s \xrightarrow{v_1} d_3 \xrightarrow{v_5} d_5 \xrightarrow{v_2} d_5 \xrightarrow{v_5} d_7 \xrightarrow{v_3} d_7 \xrightarrow{v_5} p_t)$

We use **key partition** to refer to a partition that covers the start point p_s, the terminal point p_t, or a subset of query keywords. We use $KP(\cdot)$ to denote the sequence of key partitions on a route. In Table B.2, we can find $KP(R_1) = KP(R_2) = KP(R_3) = KP(R_4) = \langle v_1, v_2, v_3, v_5 \rangle$, where the four partitions correspond to p_s, *oppo*, *costa*, and p_t, respectively. Routes R_1 to R_4 have the same start and terminal points, and they pass the four key partitions in the same order with different partial routes in between. Consequently, they are homogeneous but some of them have more complicated routing forms. To this end, we have the following two definitions.

Definition 2 (Homogeneous Routes). *Routes R_i and R_j are* **homogeneous routes** *if $R_i.head = R_j.head$, $R_i.tail = R_j.tail$, and $KP(R_i) = KP(R_j)$.*

Definition 3 (Prime Route). *Suppose HR is a complete set of homogeneous routes for a routing query, we say a route $R_i \in HR$ is* **prime** *against $R_j \in HR$ if $\delta(R_i) < \delta(R_j)$. R_i is a* **prime route** *if R_i is prime against all other routes in HR.*

By the concept of the prime route, we integrate the diversity principle into our routing query such that only prime routes should be included to ensure the diversity of search results.

Example B.2.2 (An Example of Prime Route)
Assuming R_1 to R_4 in Table B.2 are the only four regular routes from p_s to p_t having the sequence of key partitions $\langle v_1, v_2, v_3, v_5 \rangle$. Referring to the geometric depiction of Fig. B.1, we have $\delta(R_3) > \delta(R_4) > \delta(R_2) > \delta(R_1)$ and thus R_1 is the prime route among them. Consequently, only R_1 should be considered for the routing results.

Our study concentrates on finding qualified routes without exhaustive search. We define our research problem as follows.

Problem (Indoor Top-k Keyword-aware Routing Query). *Given a start point p_s, a terminal point p_t, a distance constraint Δ, and a query keyword list QW, an* **indoor top-k keyword-aware routing query** *$IKRQ(p_s, p_t, \Delta, QW, k)$ returns k regular and prime routes from p_s to p_t in a k-set Θ such that $\forall R \in \Theta$, $\delta(R) \leq \Delta$ and $\Psi(R, \Delta, QW) \geq \Psi(R', \Delta, QW)$ for any route $R' \notin \Theta$ from p_s to p_t with $\delta(R') \leq \Delta$.*

Above, $\Psi(R, \Delta, QW)$ captures the ranking score for a route R, which takes into account both spatial distance and keyword relevance for R and a given routing query. We proceed to detail the design of our ranking mechanism for routes.

B.3 Ranking Relevant Routes for IKRQ

B.3.1 Organization of Indoor Space Keywords

We differentiate two types of keywords associated with indoor partitions. An **identity word** (i-word) identifies the specific name of a partition, while a **thematic word** (t-word) [7] refers to a tag relevant to that partition. A partition can relate to one i-word only but a set of t-words. For a specific indoor venue, i-words can be obtained from floor map or the like, and t-words can be extracted from the semantic descriptions of the indoor partitions or those of the corresponding i-words. For example, i-words in a mall are shop names like *starbucks* and *zara* and function area names like *frontdesk* and *toilet*. Meanwhile, a shop *zara* can be associated with many t-words such as *pants*, *sweater* and *coat*.

Given an indoor venue, we organize its i-words and t-words in two disjoint sets. If a word is in the i-word set W_i, it is excluded from the t-word set W_t to keep the two keyword sets distinct. Given the full set V of partitions in an indoor venue, a **P2I** mapping $P2I(v_k)$ maps a partition $v_k \in V$ to its associated i-word $w_i \in W_i$, and an **I2P** mapping $I2P(w_i)$ maps an i-word $w_i \in W_i$ to a set of relevant partitions. Moreover, an

I2T mapping $I2T(w_i)$ maps an i-word $w_i \in W_i$ to a set of relevant t-words, and a **T2I** mapping $T2I(w_t)$ maps a t-word $w_t \in W_t$ to a set of relevant i-words.

In our setting, we maintain P2I as a *many-to-one* mapping and I2P as a *one-to-many* mapping such that an i-word can be associated to different partitions while a partition can only be identified by one i-word. For example, there may be five cashiers in a mall that are distributed in different partitions, but all these partitions are identified by an i-word *cashier*. Moreover, we maintain I2T and T2I as two *many-to-many* mappings, meaning that one i-word can be associated to multiple t-words and vice versa. For a partition v_k, we define its **partition words** $PW(v_k)$ as $\{P2I(v_k), I2T(P2I(v_k))\}$ that consists of an i-word $w_i = P2I(v_i)$ and a set of t-words relevant to w_i as indicated by I2T mapping. For simplicity of presentation, we assume two partitions with the same i-word have the same set of t-words.

Example B.3.1 (Examples of Keyword Mapping)
Fig. B.2 illustrates parts of the indoor space keyword mappings for the example in Fig. B.1. Partition v_3 is mapped to an i-word *costa* via the P2I mapping. Reversely, we can use the I2P mapping to find that the i-word *apple* is associated with partition v_{10}. According to the I2T and T2I mappings, t-words *laptop* and *smartphone* are relevant to the i-word *apple*, and the i-word *costa* is relevant to t-words *coffee* and *mocha*. Moreover, v_3's partition words $PW(v_3) = \{costa, \{coffee, mocha, \ldots\}\}$.

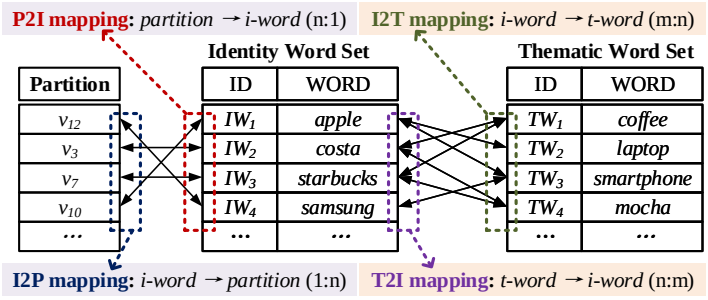


Fig. B.2: Indoor Space Keyword Mappings

In our organization, i-words act as the pivot between partitions and t-words, as the vocabulary of i-words is much smaller than that of t-words, making the mappings between i-words and partitions more concise. Using the mappings described above, we are able to quantify the keyword relevance between query keywords and routes.

B.3.2 Keyword Relevance between Query Keywords and Routes

Given a query keyword list QW, we convert each query word w_Q in QW into a set of candidate i-words for facilitating a matching between query words and partitions

(and therefore routes). If a query word w_Q is an i-word, we use the word itself as a candidate. If w_Q is a t-word, we use the T2I mapping to obtain a set of relevant i-words that are called **direct matching i-words**. However, only using direct matching i-words may lead to a very sparse candidate set. As each i-word is associated with a set of t-words that can be regarded as word features, we can measure the similarity between each direct matching i-word and other i-words, and retrieve those i-words that is highly similar to the direct matching i-words. We called such i-words **indirect matching i-words**. For example, in Fig. B.2, suppose a query word is *laptop* and we can find a direct matching i-word *apple*. Next, for *apple* we find another i-word *samsung* that shares some t-words with *apple*, e.g., both contain t-word *smartphone*. In this sense, *samsung* is similar to *apple*, and *samsung* can be obtained as an indirect matching i-word of *laptop*. By considering indirect matching i-words, we offer users more choices in routing.

Definition 4 (Candidate I-word Set). *Given a query keyword $w_Q \in QW$, its* **candidate i-word set** $\kappa(w_Q)$ *is a set of entries each of which is in form of (w_i, s), a pair of a matching i-word w_i and the similarity score s between w_Q and w_i. Two cases are discussed in deriving $\kappa(w_Q)$.*

- *If w_Q is an i-word, the matching i-word can only be w_Q itself with the similarity score 1, i.e., $\kappa(w_Q) = \{(w_Q, 1)\}$.*
- *If w_Q is a t-word, the matching i-word(s) should include*
  - *each direct matching i-word $w_i' \in T2I(w_Q)$ with the similarity score 1, denoted as $(w_i', 1)$.*
  - *each indirect matching i-word w_i'' such that $I2T(w_i'') \cap \bigcup_{w_i \in T2I(w_Q)} I2T(w_i) \neq \varnothing$, where $\bigcup_{w_i \in T2I(w_Q)} I2T(w_i)$ is the union set of the t-words of each i-word in $T2I(w_Q)$. In such a case, the similarity is measured in the form of* Jaccard Similarity *as $s(w_i'') = \frac{|I2T(w_i'') \cap \bigcup_{w_i \in T2I(w_Q)} I2T(w_i)|}{|I2T(w_i'') \cup \bigcup_{w_i \in T2I(w_Q)} I2T(w_i)|}$.*

To avoid long tails, we only keep the entries whose similarity scores are greater than a certain threshold τ in $\kappa(w_Q)$. We use $\kappa(w_Q).W_i$ to denote the set of matching i-words in $\kappa(w_Q)$.

Corresponding to Fig. B.2, some partitions and their partition words are listed below.

partition	i-word	t-words
v_3	*costa*	$\{coffee, drinks, macha\}$
v_{10}	*apple*	$\{phone, mac, laptop, watch\}$
v_7	*starbucks*	$\{coffee, macha, latte, drinks\}$
v_{12}	*samsung*	$\{phone, laptop, earphone\}$

Example B.3.2 (An Example of Candidate I-word Set)
Given a query keyword list $QW = \langle latte, apple \rangle$, we set $\tau = 0.5$. As keyword *latte* is a t-word, we have $T2I(latte) = \{starbucks\}$. Thus, $(starbucks, 1)$ is included as a candidate i-word since *starbucks* is a direct matching of *latte*. Furthermore, as i-word *costa* that is not a direct matching of *latte*, we have $s(costa) = \frac{|I2T(costa) \cap \bigcup_{w_i \in T2I(latte)} I2T(w_i)|}{|I2T(costa) \cup \bigcup_{w_i \in T2I(latte)} I2T(w_i)|}$. Since $I2T(costa) = \{coffee, drinks, macha\}$ and $\bigcup_{w_i \in T2I(latte)} I2T(w_i) = \{coffee, drinks, macha, latte\}$, we have $s(costa) = 3/4 = 0.75$. Likewise, we have $s(apple) = s(samsung) = 0$. Consequently, $\kappa(latte) = \{(starbucks, 1), (costa, 0.75)\}$ and $\kappa(latte).W_i = \{starbucks, costa\}$. As the other keyword *apple* is an i-word, we have $\kappa(apple) = \{(apple, 1)\}$ and $\kappa(apple).W_i = \{apple\}$. Finally, the original query keyword list can be converted to a list of candidate i-word sets: $\mathbf{K}(QW) = \langle \kappa(latte), \kappa(apple) \rangle = \langle \{(starbucks, 1), (costa, 0.75)\}, \{(apple, 1)\} \rangle$.

On the other hand, given an item x on a route R, we use an operator $v_*(x)$ to obtain its relevant partitions. If x is a door, we obtain all the partitions that one can leave through door x, i.e., $v_*(x) = D2P_{\sqsubset}(x)$. Otherwise, x is a point and we obtain the partition that contains point x, i.e., $v_*(x) = v(x)$. Accordingly, we look at i-words in a route.

Definition 5 (Route Words). *Given $R = (x_s, d_i, \ldots, d_n, x_t)$, its* **route words** *are the union of all its relevant partitions' associated i-words, computed as $RW(R) = \bigcup_{x \in R} PW(v_*(x)).w_i$.*

Example B.3.3 (Examples of Route Words)
Referring to the route $R = (p_s, d_3, p_t)$ in Fig. B.1, for point p_s, we have $PW(v(p_s)).w_i = PW(v_1).w_i = \{zara\}$. Likewise, we have $PW(v(p_t)).w_i = \varnothing$. For door d_3, we have $PW(D2P_{\sqsubset}(d_3)).w_i = PW(v_1).w_i \cup PW(v_5).w_i = \{zara\} \cup \varnothing = \{zara\}$. Consequently, we have $RW(R) = \{zara\}$.

Next, route R's keyword relevance is defined as follows.

Definition 6 (Keyword Relevance). *Given a query keyword list QW, a route R's* **keyword relevance** *w.r.t. QW is*

$$\rho_{QW}(R) = \begin{cases} 0, & \text{if } N_{QW}(R) = 0; \\ N_{QW}(R) + \frac{\sum_{w_Q \in QW} \left(\max_{w'_i \in \mathrm{M}(w_Q, R)} s(w'_i) \right)}{N_{QW}(R)}, & \text{otherwise.} \end{cases}$$

Above, $N_{QW}(R)$ is the number of R's route words that are relevant to query words in QW, and $\mathrm{M}(w_Q,R) = \kappa(w_Q).W_i \cap RW(R)$ denotes the set of w_Q's matching i-words on R. When the context is clear, we use $\rho(R)$ to denote the keyword relevance. Also, $\rho(R)$ is positive if there is at least one query keyword covered by R (i.e., $N_{QW}(R) > 0$). In such a case, the left part of the summation indicates the number of query keywords that R covers, and the right part measures the average of each covered keyword w_Q's *maximum* similarity score with its matching i-words on R (i.e., $\mathrm{M}(w_Q,R)$). In the best case, all keywords in QW can match an i-word on R with similarity score 1. Thus, the range of $\rho(R)$ is $0 \cup (1, |QW|+1]$. The computation complexity of $\rho(R)$ is $O(mn)$, where m is $|QW|$ and n is the number of i-words in a route R.

Example B.3.4 (Examples of Keyword Relevance)
Referring to Fig. B.1, we have two routes $R_1 = (p_s, d_2, d_5, d_7, d_7, p_t)$ and $R_2 = (d_{15}, d_{15}, d_{14}, d_{13}, d_7, d_6)$ and a query keyword list $QW = \{latte, apple\}$. For route R_1, we have $RW(R_1) = \{zara, oppo, costa\}$, and $\kappa(latte).W_i \cap RW(R_1) = \{costa\}$ with similarity score 0.75 and $\kappa(apple).W_i \cap RW(R_1) = \varnothing$, thus $\rho(R_1) = 1 + \frac{0.75}{1} = 1.75$.

For route R_2, we have $RW(R_2) = \{apple, starbucks, costa\}$, $\kappa(latte).W_i \cap RW(R_2) = \{starbucks, costa\}$, and $\kappa(apple).W_i \cap RW(R_2) = \{apple\}$. Consider the two candidate i-words of query keyword *latte*, we select i-word *starbucks* with the maximum similarity score as $s(starbucks) = 1 > s(costa) = 0.75$. Consequently, $\rho(R_2) = 2 + \frac{1+1}{2} = 3$.

B.3.3 Ranking Score for Routes

Definition 7 (Ranking Score). *Given a query IKRQ(p_s, p_t, Δ, QW, k) and a route R from p_s to p_t, the **ranking score** of R is computed as a linear combination of the normalized scores of keyword relevance and spatial relevance as follows.*

$$\psi(R, \Delta, QW) = \alpha \cdot \frac{\rho(R)}{|QW|+1} + (1-\alpha) \cdot \left(\frac{\Delta - \delta(R)}{\Delta}\right) \tag{B.1}$$

We use $\psi(R)$ for simplicity when the context is clear. Our ranking score can be flexibly customized by the tradeoff parameter $\alpha \in [0,1]$ according to specific application needs [8–11]. For example, a shopper in a mall may prefer the routes covering the query keywords as much as possible for the sake of shopping, and the requirement for walking distance can be less important. In this case, a large α can boost the keyword score. In contrast, passengers in airports are often more sensitive to distance constraints and would accept some query keywords missing. Thus, a small α can be used to emphasize the distance. The effect of α is studied experimentally.

B.4 Search Algorithms for IKRQ

A naive idea for our routing works as follows. We iteratively find candidate partial routes from the start point, validate them using the distance constraint and the two principles (Section B.2.2), and expand them through doors. After all complete routes have been seen, we return the k routes with the highest ranking scores. This method is inefficient as it finds all complete routes through expensive expansions. To improve the efficiency, we can identify unpromising route branches and avoid expanding to them, which is enabled by pruning rules.

B.4.1 Pruning Rules for Expansion

Our pruning rules use the skeleton distance [12] as the *lower bound indoor distance* for two indoor items x_i and x_j.

$$|x_i,x_j|_L = \begin{cases} |x_i,x_j|_E, & \text{if } x_i \text{ and } x_j \text{ are on the same floor;} \\ \min\limits_{sd_i \in SD(x_i), sd_j \in SD(x_j)} \big(|x_i,sd_i|_E + \delta_{s2s}(sd_i,sd_j) + \\ |sd_j,x_j|_E\big), & \text{otherwise.} \end{cases}$$

Specifically, $|x_i,x_j|_L$ is the Euclidean distance if x_i and x_j are on the same floor. Otherwise, one needs to go through a number of staircase doors (e.g., $sd_i \in SD(x_i)$) to reach x_j from x_i, and there can be multiple such paths. In this case, $|x_i,x_j|_L$ is the shortest path distance among all such paths.

With the lower bound distance, we derive the following pruning rules for a query IKRQ(p_s,p_t,Δ,QW,k).

Pruning Rule 6. *A partial route $R^\star = (p_s,d_i,\ldots,d_n)$ in the searching can be pruned if $\delta(R^\star) + |d_n,p_t|_L > \Delta$.*

Pruning Rule 7. *A door d_n can be pruned out of the search if $|p_s,d_n|_L + |d_n,p_t|_L > \Delta$.*

Pruning Rule 8. *An indoor partition v_i can be pruned out of the search if its lower bound distance $\delta(p_s,v_i,p_t)$ =*

$$\min_{d_i \in P2D_\sqsupset(v_i), d_j \in P2D_\sqsubset(v_i)} (|p_s,d_i|_L + \delta_{d2d}(d_i,d_j) + |d_j,p_t|_L) > \Delta.$$

Example B.4.1 (Examples of Pruning Rule 8)
Referring to Fig. B.1, suppose we need to route from p_s to p_t with the distance constraint Δ = 16m, and we have obtained a partial route $R^\star = (p_s,d_2,d_5)$ whose distance is 12.5m. $R^\star$ can be pruned according to Pruning Rule 6, since $\delta(R^\star) + |d_5,p_t|_E$ = 12.5m + 6m $> \Delta$ = 16m. Also, suppose that $|p_s,d_6|_E + |d_6,p_t|_E$ = 13m + 5m $> \Delta$ = 16m, door d_6 can be discarded according to Pruning Rule 7. Take a close

look at partition v_3 whose only two doors are d_6 and d_7, and $|p_s, d_6|_E$ + $|d_6, d_7|_E$ + $|d_7, p_t|_E$ = 13m + 4.5m + 1m > Δ = 16m, v_3 can also be discarded according to Pruning Rule 8.

Furthermore, we derive the upper bound of the ranking score to enable the following pruning rule.

Pruning Rule 9. *Given the current k-th highest ranking score ψ_k among the seen complete routes, a partial route $R^\star = (p_s, d_i, \ldots, d_n)$ can be pruned if its upper bound ranking score $\psi_U(R^\star) = \alpha \cdot 1 + (1-\alpha)(1 - (\delta(R^\star) + |d_n, p_t|_L)/\Delta) \leq \psi_k$.*

In Pruning Rule 9, we upper bound a partial route's final ranking score by an overestimate of its keyword and spatial scores. The former is overestimated to 1 as a full coverage of query keywords and the latter is computed based on the lower bound indoor distance to p_t (i.e., $\delta(R^\star) + |d_n, p_t|_L$). This pruning rule enables the *k*bound pruning, i.e., we can discard a partial route if its upper bound ranking score is not higher than the *k*-th best score among the routes already obtained.

Furthermore, recall that only a prime route should be returned among all homogeneous routes (see the diversity principle in Section B.2.2). To this end, we have the following lemma and a corresponding pruning rule.

Lemma 1. *Given a query IKRQ(p_s, p_t, Δ, QW, k), if route $R = (p_s, d_i, \ldots, d_n, p_t)$ is a returned prime route, each of its partial route $R^\star = (p_s, \ldots, d_k)$ ($i \leq k \leq n$) is also a prime route.*

Proof. *Without loss of generality, we represent R's remaining route that continues with $R^\star$ as $R^\vdash = (d_k, \ldots, p_t)$. We prove the lemma by contradiction. Suppose that $R^\star$ is not a prime route and $R^{\star'}$ is $R^\star$'s homogeneous route such that $\delta(R^{\star'}) < \delta(R^\star)$ and $KP(R^{\star'}) = KP(R^\star)$. According to Definition 2, we have $R^{\star'}.tail = R^\star.tail = d_k$. By concatenating $R^{\star'}$ and $R^\vdash$ at d_k, we can obtain a complete route R' from p_s to p_t. Moreover, we have its sequence of key partitions $KP(R') = concat(KP(R^{\star'}), KP(R^\vdash)) = concat(KP(R^\star), KP(R^\vdash)) = KP(R)$. According to Definitions 2 and 3, we find R' is a homogeneous route of R and is prime against R. Thus, R cannot be a prime route.* □

According to Lemma 1, a partial route cannot be a prime route if the search has already found a homogeneous route that is prime against it. This enables the following pruning rule.

Pruning Rule 10. *A partial route $R^\star = (p_s, d_i, \ldots, d_n)$ in the search can be pruned if the search has already obtained a route $R^{\star'}$ from p_s to d_n that is prime against $R^\star$.*

Combining the definition of the prime route with the regularity principle in Section B.2.2, we have the following lemma.

Lemma 2. *Given a route R returned by the IKRQ(p_s, p_t, Δ, QW, k), R can have a loop of two consecutive identical doors (d_k, d_k) only if the loop passes a key partition that covers at least one query keyword in QW.*

Proof. *(Sketch) We prove it by contradiction. Suppose that $R = (p_s, \ldots, d_k, d_k, \ldots, p_t)$ contains a loop (d_k, d_k) that does not pass any key partition. According to Definition 3, there must be a homogeneous route $R' = (p_s, \ldots, d_k, \ldots, p_t)$ of R and R' is prime against R in that $KP(R') = KP(R)$ and $\delta(R') < \delta(R)$. This violates the diversity principle.* □

Example B.4.2 (Examples of Lemma 2)
Suppose IKRQ(p_s, p_t, 25m, {*latte*, *apple*}, 1) is issued in the setting shown in Fig. B.1, the parameter α is 0.2, and the search has obtained a complete route $R_1 = (p_s, d_2, d_6, d_7, p_t)$ whose distance is 20m. According to Example B.3.4, we have R_1's keyword relevance is 1.75 and its ranking score is $0.2 \cdot \frac{1.75}{3} + 0.8 \cdot \frac{25-20}{25}$ = 0.277 according to Equation B.1. Thus, the current *k*bound is updated to 0.277. Next, the search expands to a route $R_2^\star = (p_s, d_2, d_5, d_7, d_7)$ whose distance is 22.5m and lower bound distance to p_t is 22.5m + $|d_7, p_t|_E$ = 22.5m + 1m = 23.5m. According to Pruning Rule 9, $R_2^\star$'s ranking score is upper-bounded by $0.2 \cdot 1 + 0.8 \cdot \frac{25-23.5}{25}$ = 0.248. So $R_2^\star$ should be pruned as its upper bound ranking score is smaller than the current *k*bound.

Suppose that two partial routes have been obtained between p_s and d_5, namely $R_3^\star = (p_s, d_2, d_5)$ and $R_4^\star = (p_s, d_3, d_5, d_5)$. Both routes pass a key partition v_2 (its i-word *oppo* is an indirect matching of *apple*) and we have $\delta(R_3^\star)$ = 12.5m and $\delta(R_4^\star)$ = 23.2m. Currently, $R_3^\star$ is prime against $R_4^\star$ and therefore $R_4^\star$ should be pruned according to Pruning Rule 10. Moreover, according to Lemma 2, when the search has expanded to a door d_9, the next hop cannot be d_9 again as neither of its relevant partitions (v_5 and v_6) covers a query keyword.

B.4.2 Overall Search Framework

Based on the above pruning rules and lemmas, we formalize our overall framework in Algorithm 1. A priority queue Q (initialized in line 1) is used to control the order of route expansion. The local information of the current expansion is kept in a five-tuple *stamp* $S(v, R, \delta, \rho, \psi)$, where R is a route that has been expanded to a door or the terminal point so far, v is the last partition that R reaches, and δ, ρ, ψ are R's route distance, keyword relevance, and ranking score, respectively. The architecture of our search algorithms is depicted in Fig. B.3.

The initialization (lines 1–6) obtains a set W_{ci} of all candidate i-words w.r.t. query keyword list QW (line 2), and computes a set P of all key partitions covering at least one keyword in QW (line 3). We exclude the partition $v(p_s)$ from P and add the

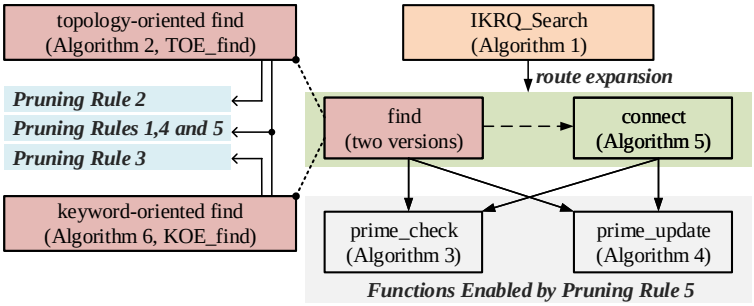


Fig. B.3: Architecture of the IKRQ Search Algorithms

partition $v(p_t)$ to P to regularize the route search. Sets D_f and D_n hold the doors already explored (line 4). Doors in D_f are filtered by Pruning Rule 7, whereas those in D_n are not.

Algorithm 1 IKRQ_Search (p_s, p_t, Δ, QW, k)

1: initialize priority queue Q
2: set of all candidate i-words $W_{ci} \leftarrow \bigcup_{w_Q \in QW} \kappa(w_Q).W_i$
3: $P \leftarrow \left(\bigcup_{w_Q \in QW} I2P(\kappa(w_Q).W_i)\right) \setminus v(p_s) \cup v(p_t)$
4: door sets $D_n \leftarrow \varnothing$, $D_f \leftarrow \varnothing$
5: $kbound \leftarrow 0$
6: initialize hashtable H_{prime}
7: $R_0 \leftarrow (p_s)$
8: $S_0 \leftarrow (v(p_s), R_0, 0, \rho(R_0), \psi(R_0))$
9: Q.$push(S_0)$
10: **while** Q is not empty **do**
11: $S_i \leftarrow$ Q.$pop()$
12: $ES \leftarrow$ `find`(S_i) ▷ find the next valid stamps
13: **for** each $S_j \in ES$ **do**
14: `connect`(S_j) ▷ connect each valid stamp to terminal
15: **return** current top-k results

Subsequent routing skips doors in D_f, and exempts doors in D_n from repeated checks by Pruning Rule 7. We initialize the *kbound* for Pruning Rule 9 (line 5), and a hashtable H_{prime} to store the route temporarily prime against others for Pruning Rule 10 (line 6). The algorithm then performs the expansion iteratively (lines 7–14). It generates an initiate route (p_s) and its corresponding stamp S_0 (lines 7–8). Next, it pushes S_0 into Q and iterates on Q until all stamps have been expanded to p_t (lines 9–14). The search follows a *find-and-connect* paradigm. That is, in each iteration, it fetches a stamp S_i with the highest ranking score from Q (line 11), expands the current stamp to find a set *ES* of valid stamps based on the pruning rules (calling function

`find`() in line 12), and attempts to connect each valid stamp in *ES* to the destination if some condition is met (calling function `connect`() in line 14). The top-k results are returned when Q is empty (lines 10 and 15).

Given a valid stamp S_i, we propose two versions of strategies to find the next valid stamps. One is based on the indoor topology information and the other is based on the query keywords. The search algorithms using the two different strategies are called topology-oriented expansion (ToE) and keyword-oriented expansion (KoE), respectively. Function `find`() is instantiated as `ToE_find`() and `KoE_find`(), respectively.

B.4.3 Topology-oriented Expansion (ToE)

The idea of `find`() in ToE is to reach all accessible doors from the current door based on indoor topology. We formalize this strategy in Algorithm 2.

Algorithm 2 `ToE_find` (Stamp S_i)

```
1:  set ES ← ∅
2:  (v_i, R_i, δ_i, ρ_i, ψ_i) ← S_i; d_k ← R_i.tail
3:  if prime_check(S_i, H_prime) is false then return                ▷ Pruning Rule 10
4:  for each d_l in P2D_□(v_i) \ D_f do
5:      if d_l ∈ R_i and d_l ≠ R_i.tail then continue                ▷ regularity check
6:      if d_l ∉ D_n then                                            ▷ Pruning Rule 7
7:          if |p_s, d_l|_L + |d_l, p_t|_L > Δ then
8:              D_f ← D_f ∪ d_l; continue
9:          else
10:             D_n ← D_n ∪ d_l
11:     v_j ← D2P_□(d_l) \ v_i
12:     if d_k == d_l and PW(v_i).w_i ∉ W_ci then
13:         continue                  ▷ regularity check based on Lemma 2
14:     if δ_i + δ_d2d(d_k, d_l) > Δ then continue                   ▷ distance constraint check
15:     δ_LB ← δ_i + δ_d2d(d_k, d_l) + |d_l, p_t|_L
16:     if δ_LB > Δ then continue                                    ▷ Pruning Rule 6
17:     ψ_UB ← α · 1 + (1 − α)(1 − δ_LB / Δ)
18:     if ψ_UB ≤ kbound then continue                               ▷ Pruning Rule 9
19:     R_j ← append d_l to R_i
20:     S_j ← (v_j, R_j, δ(R_j), ρ(R_j), ψ(R_j))
21:     prime_update(S_j, H_prime)
22:     add S_j to ES
23: return ES
```

In particular, line 1 initializes a set *ES* to save the valid stamps to be found, and line 2 obtains the current stamp S_i and the current door d_k from the tail of the corresponding route R_i. To determine if S_i is a temporary prime route that does not need

to be pruned (c.f. Pruning Rule 10), ToE calls a function `prime_check`() to compare S_i's route R_i to its homogeneous routes already recorded in a global hashtable H_{prime} (line 3).

The function `prime_check`() is detailed in Algorithm 3. First, the key for identifying R_i's homogeneous routes is formed as $(R_i.tail, KP(R_i))$, a pair of R_i's tail door and R_i's sequence of key partitions[3] (line 2). The function returns *true* if the shortest distance among all homogeneous routes in H_{prime} does not exist or is greater than R_i's distance δ_i. Otherwise, it returns *false* to indicate that R_i is not the temporary prime route and should be pruned.

Algorithm 3 `prime_check` (Stamp S_i, Hashtable H_{prime})

1: $(v_i, R_i, \delta_i, \rho_i, \psi_i) \leftarrow S_i$
2: $key \leftarrow (R_i.tail, KP(R_i))$
3: **if** $\mathsf{H}_{prime}[key] = \varnothing$ or $\mathsf{H}_{prime}[key] > \delta_i$ **then return** *true* **else return** *false*

Back to line 4 in Algorithm 2, ToE tests on each leavable door d_l of R_i's last reached partition v_i. It excludes those doors in the global set D_f that have been pruned by Pruning Rule 7. Before applying Pruning Rule 7, line 5 performs a regularity check. Specifically, if d_l has been visited by R_i before ($d_l \in R_i$), it can be the next door only when R_i's last visited door is also d_l (a loop within one-hop in regularity principle). Hence, d_l should be pruned if $d_l \neq R_i.tail$. Afterwards, ToE examines d_l based on Pruning Rule 7. Specifically, if d_l is not in D_n (line 6), ToE computes the lower bound distance w.r.t. d_l (line 7). If it exceeds Δ, ToE adds it to D_f to make sure it is not processed in subsequent routing. Otherwise, ToE adds it to D_n.

Next, ToE performs checks according to the query principles and pruning rules. Particularly, lines 11-13 check the regularity for two identical doors according to Lemma 2, in which v_j is the partition that connects the d_k and d_l on the route. Line 14 checks the distance constraint for the route to be expanded to d_l, and lines 15-16 further derive its lower bound and verify it according to Pruning Rule 6. In the end, ToE uses Pruning Rule 9 to remove the expansion whose derived upper bound ranking score cannot exceed the *kbound* of the search (lines 17-18). Once the check is done, ToE validates the expansion to d_l by appending d_l to the end of R_j and generating the corresponding stamp S_j (lines 19-20). Moreover, it calls function `prime_update`() to update the temporary prime route with S_j (line 19). When each accessible door d_l has been explored, `ToE_find`() returns the set *ES* that contains all valid stamps.

Algorithm 4 `prime_update` (Stamp S_i, Hashtable H_{prime})

1: $(v_i, R_i, \delta_i, \rho_i, \psi_i) \leftarrow S_i$
2: $key \leftarrow (R_i.tail, KP(R_i))$
3: **if** $\mathsf{H}_{prime}[key] = \varnothing$ or $\mathsf{H}_{prime}[key] > \delta_i$ **then** $\mathsf{H}_{prime}[key] \leftarrow \delta_i$

[3] In our routing, all expanding routes have the same head item, i.e., p_s.

Algorithm 4 formalizes the function `prime_update`(). The hash key generation is the same as its counterpart of `prime_check`(). Their difference is that `prime_update` puts the distance of the route R_i into H_{prime} if R_i is currently prime against its homogeneous routes.

We proceed to present how to connect each valid stamp returned by `ToE_find`(). The process is formalized in Algorithm 5.

Algorithm 5 `connect` (Stamp S_j)

1: $(v_j, R_j, \delta(R_j), \rho(R_j), \psi(R_j)) \leftarrow S_j$
2: **if** $v_j == v(p_t)$ **then** ▷ reach a door in the same partition with p_t
3: $R_f \leftarrow$ append p_t to R_j
4: $S_f \leftarrow (v(p_t), R_f, \delta(R_f), \rho(R_f), \psi(R_f))$
5: **if** $\delta(R_f) \leq \Delta$ and $\psi(R_f) > kbound$ and `prime_check`(S_f, H_{prime}) is *true* **then**
6: update top-k results and *kbound* with R_f
7: `prime_update`(S_f, H_{prime})
8: **else**
9: **if** `prime_check`(S_j, H_{prime}) is *false* **then**
10: **continue** ▷ Pruning Rule 10
11: **if** $\rho(R_j) = |QW| + 1$ **then** ▷ all keywords has been covered
12: find shortest *regular* route $(d_j, d_x, \ldots, p_t)$ ▷ regularity check
13: $R_f \leftarrow$ append $(d_x, \ldots, p_t)$ to R_j
14: $S_f \leftarrow (v(p_t), R_f, \delta(R_f), \rho(R_f), \psi(R_f))$
15: **if** $\delta(R_f) \leq \Delta$ and $\psi(R_f) > kbound$ and `prime_check`(S_f, H_{prime}) is *true* **then**
16: update top-k results and *kbound* with R_f
17: `prime_update`(S_f, H_{prime})
18: **else** ▷ can be further expanded
19: Q.*push*(S_j)

For stamp S_j to be connected, we first determine if it has reached the same partition of the terminal point p_t (line 2). If so, we immediately connect the end of the corresponding route R_j to p_t and check if the resulting route R_f meets the query conditions (lines 3–5). If so, we add R_f to the top-k results and update the current *kbound* (lines 6–7). Otherwise, we explore how S_j can be further processed (lines 8–19). Here we call `prime_check`() again to verify if S_j holds the temporary prime route (lines 9–10). Afterwards, we check if the current route R_j has already covered all the query keywords (line 11). If so, there is no necessary to reach any other key partitions. Therefore, we immediately connect the end of R_j to p_t by finding a shortest regular route[4], and obtain a final stamp S_f (lines 12–14). Afterwards, we add the qualified route R_f to the top-k results and update the current *kbound* (lines 15–17). If R_j does not cover all the query keywords, we push S_j into the queue for further expan-

[4]Note that a global regularity check is required when connecting R_j to p_t.

sion (lines 18–19). Lines 2–17 in Algorithm 5 utilize a heuristic rule that the current stamp should connect to the destination directly when a certain condition is met, i.e., it has reached the destination partition or covered all query keywords. As a result, the *kbound* and prime routes are updated as soon as possible, which in turn help to prune more aggressively.

B.4.4 Keyword-oriented Expansion (KoE)

ToE always expands from the current door to the next enterable door within one hop. However, such one-hop expansions cannot guarantee covering some query keyword(s). An alternative is to focus on the query words that have not been covered by the current stamp, and directly expand to one of the key partitions that can cover some of those uncovered query words. This idea is called keyword-oriented expansion (KoE), and its finding strategy is formalized in Algorithm 6.

Algorithm 6 `KoE_find` (Stamp S_i)

1: set $ES \leftarrow \varnothing$
2: $(v_i, R_i, \delta_i, \rho_i, \psi_i) \leftarrow S_i$; $d_k \leftarrow R_i.tail$
3: **if** `prime_check`(S_i, H_{prime}) is *false* **then return** ▷ Pruning Rule 10
4: $P' \leftarrow P$ ▷ find candidate key partitions
5: **for** $w_Q \in QW$ **do**
6: **if** $\kappa(w_Q).W_i \cap RW(R_i) \neq \varnothing$ and $d_k \neq p_s$ **then**
7: $P' \leftarrow P' \setminus I2P(\kappa(w_Q).W_i)$
8: **for** v_j in P' **do**
9: **if** $\delta_{LB}(p_s, v_j, p_t) > \Delta$ **then**
10: $P \leftarrow P \setminus v_j$; **continue** ▷ Pruning Rule 8
11: **if** $\delta_i + \delta_{LB}(d_k, v_j, p_t) > \Delta$ **then continue** ▷ distance constraint check
12: **for** each $d_x \in P2D_{\sqsubset}(v_i)$ and $d_l \in P2D_{\sqsupset}(v_j)$ **do**
13: find shortest *regular* route $(d_k, d_x, \ldots, d_l)$ ▷ regularity check
14: $R_j \leftarrow$ append $(d_x, \ldots, d_l)$ to R_i
15: $\delta_{LB} \leftarrow \delta(R_j) + |d_l, p_t|_L$
16: **if** $\delta_{LB} > \Delta$ **then continue** ▷ Pruning Rule 6
17: $\psi_{UB} \leftarrow \alpha \cdot 1 + (1-\alpha)(1 - \delta_{LB}/\Delta)$
18: **if** $\psi_{UB} \leq kbound$ **then continue** ▷ Pruning Rule 9
19: $S_j \leftarrow (v_j, R_j, \delta(R_j), \rho(R_j), \psi(R_j))$
20: `prime_update`(S_j, H_{prime})
21: **return** ES

The processing on the current stamp S_i (lines 1–3) is the same as the counterpart in Algorithm 2. It is noteworthy that here v_i must be a key partition and d_k must be an enterable door of v_i since in each expansion KoE has to reach a key partition. Next, unlike ToE that iterates on each enterable door based on indoor topology,

KoE searches for the candidate partitions relevant to the uncovered query keywords (lines 4–7). Specifically, it copies the key partition set P (initialized in line 2 of Algorithm 1) to a local set P' (line 4), iterates on each query word $w_Q \in QW$, and checks if w_Q has been covered by the current route R_i in S_i (lines 5–6). If so, its corresponding key partitions should be removed from P' (line 7). In line 6, a case is handled separately. When the initial stamp S_0 is encountered ($d_k = p_s$), we do not remove any partition from P'. This ensures that no extra constraint on partitions is added.

Afterwards, KoE deals with each candidate partition $v_j \in P'$ to find a route that can reach one of the enterable doors of v_j. For each candidate partition v_j, KoE derives the lower bound distance and checks it against Pruning Rule 8 (lines 9–10). If a partition v_j should be pruned, it is excluded from the global set P and never processed in subsequent expansions. Furthermore, KoE checks the distance constraint for the routes to be expanded to the doors of v_j, whose lower bound distance is computed as $\delta_i + \delta_{LB}(d_k, v_j, p_t)$ (line 11). Referring to Pruning Rule 8, $\delta_{LB}(x_s, v_i, x_t)$ means the minimum indoor distance from x_s, through partition v_i, to x_t.

When v_j becomes the next target partition to reach, KoE needs to find a route from the current door d_k through a leavable door d_x in current partition v_i to an enterable door d_l in the next partition v_j. For each such combination of d_k, d_x and d_l, we may find a large number of qualified routes. However, the following lemma tells that we only need to consider the one with the shortest distance in the expansion.

Lemma 3. *Given $R_p = (p_s, \ldots, d_i, d_{i+1}, \ldots, d_j, \ldots, p_t)$ as a prime route such that d_i and d_j refer to an enterable door of two consecutive key partitions v_m and $v_{m+1} \in KP(R_p)$, respectively, and d_{i+1} refers to a leavable door of v_m. R_p's partial route $R_p^\star = (d_i, d_{i+1}, \ldots, d_j)$ must also be a prime route.*

Proof. *(Sketch) We prove it by contradiction. Suppose R_p's key partition sequence is $\langle v_s, \ldots, v_e \rangle$. We segment R_p into three partial routes: $R_p^\dashv = (p_s, \ldots, d_i)$, $R_p^\star = (d_i, d_{i+1}, \ldots, d_j)$, and $R_p^\vdash = (d_j, \ldots, p_t)$. The key partition sequences are $\langle v_s, \ldots, v_{m-1} \rangle$, $\langle v_m \rangle$, $\langle v_{m+1}, v_s \rangle$, respectively. If $R_p^\star$ is not a prime route, there must be a route $R_p^{\star'}$ having $\delta(R_p^{\star'}) < \delta(R_p^\star)$ and $KP(R_p^{\star'}) = KP(R_p^\star) = \langle v_m \rangle$. By concatenating $R_p^\dashv$, $R_p^{\star'}$, and $R_p^\vdash$, we get a route R'_p that has $KP(R'_p) = KP(R_p) = \langle v_s, \ldots, v_e \rangle$ and $\delta(R'_p) < \delta(R_p)$. Thus, R_p is not a prime route.* □

Lemma 3 can be easily extended to the situation where global regularity needs to be considered for the whole route. Therefore, given any combination of $d_k \in P2D_\sqsupset(v_i)$, $d_x \in P2D_\sqsubset(v_i)$ and $d_l \in P2D_\sqsupset(v_j)$, we only need to find the shortest route $(d_k, d_x, \ldots, d_l)$ with a regularity check (lines 12–13 in Algorithm 6). When each such route has been expanded to d_l, we generate a new route R_j and check it based on Pruning Rule 6 and 9 (lines 14–18). If those rules fail to prune anything, we form a new stamp S_j and call `prime_update()` (lines 19–20). When each candidate partition v_j has been explored, `KoE_find()` returns the set ES that contains all valid stamps. Recall that such stamps will be processed in the search framework (lines 13–14 in Algorithm 1) where the use of `connect()` is the same as that in the search of ToE.

B.5 Experimental Studies

We experimentally evaluate ToE, KoE and their variants. Table B.3 lists all routing algorithms in comparison. Specifically, ToE\D and KoE\D involve no pruning rule based on the distance constraint Δ, i.e., Pruning Rules 6, 7 and 8. ToE\B and KoE\B skip the *k*bound-based Pruning Rule 9. ToE\P skips the prime-based Pruning Rule 10. This variant does not apply to KoE, since it is formulated based on prime routes. Instead, we design KoE* that precomputes the shortest route between any two doors, which may speed up routing to the next key partition in KoE (line 13 in Algorithm 6). Note that such a route should be re-computed when the regularity check fails. All algorithms are implemented in Java and run on a PC with a 2.30GHz Intel i5 CPU and 16 GB memory.

Table B.3: Notations of Comparable Methods

Modification	ToE family	KoE family
–	ToE	KoE
no distance-based Pruning Rules 6-8	ToE\D	KoE\D
no kbound-based Pruning Rule 9	ToE\B	KoE\B
no prime-based Pruning Rule 10	ToE\P	–
with precomputed shortest routes	–	KoE*

B.5.1 Results on Synthetic Data

Settings

Indoor Space. Based on a real-world floorplan[5], we generate a multi-floor indoor space where each floor takes 1368m $\times$ 1368m with 96 rooms, 4 hallways, and 4 staircases. The irregular hallways are decomposed into smaller but regular partitions. As a result, we obtain 141 partitions and 220 doors on each floor. We duplicate the floorplan 3, 5, 7, or 9 times to simulate different indoor spaces. The four staircases of each two adjacent floors are connected by stairways, each being 20m long. In the default setting, we use a 5-floor indoor space with 705 partitions and 1100 doors.

Indoor Keywords. We assign keywords to the 96 rooms on each floor as follows. We use Scrapy[6] to crawl the online shop information from five shopping malls[7] in Hong Kong, obtaining 2074 documents for 1225 shop brands. All the 1225 brand names are used as i-words. They are then fed into the *RAKE* algorithm [13] to extract corresponding keywords from the documents. Only 1120 i-words yield extracted keywords. For each such i-word, we use up to 60 extracted keywords with the highest

[5] deviantart.com/mjponso/art/Floor-Plan-for-a-Shopping-Mall-86396406
[6] https://scrapy.org/
[7] Refer to https://longaspire.github.io/s/hkdata.html for the details.

TF-IDF values as its t-words. In total, we have 9195 t-words and each i-word corresponds to 16.6 t-words on average. We randomly assign an i-word and all its t-words to each room. The indoor space keyword mappings are of approximately 4 MB and thus kept in main memory.

Queries. For a valid IKRQ(p_s, p_t, Δ, QW, k), the distance constraint Δ must be larger than the indoor distance δ_{s2t} between p_s and p_t. Thus, we generate p_s, p_t, and Δ in the following steps. 1) We fix δ_{s2t} to a certain value and randomly select a point p_s in the space. 2) We find a door d' whose distance to p_s approximates δ_{s2t} based on the precomputed door-to-door matrix. 3) We expand from d' to find a random point p_t whose distance to p_s just meets δ_{s2t}. 4) We generate $\Delta = \eta \cdot \delta_{s2t}$, where $\eta > 1$ is a coefficient. Subsequently, we randomly select a set of keywords from the 1120 i-words and 9195 t-words to form QW. A parameter β controls the fraction of i-words in QW. The query keyword set size $|QW|$ is varied from 1 to 5, as an analysis [14] discloses that nearly all map queries contain at most 5 keywords, and statistics[8] show that 65 percent of web searchers use 1 or 2 keywords and over 94 percent of web searchers use at most 4 keywords. In addition, we also vary the tradeoff parameter α in the ranking score (c.f. Equation B.1) and the similarity threshold τ (see Definition 4). Table B.4 gives the parameter settings with default values shown in bold. It is noteworthy that users do not have to specify all of these parameters. For example, users do not need to give i-words and t-words separately. Rather, they are recognized automatically in our implementation.

Performance Metrics. We generate ten query instances with random QWs for each parameter setting. We run each instance five times, and measure the *average running time* and *average memory cost* per run of a single query instance.

Table B.4: Parameter Settings

Parameters	Settings
k	1, ..., **7**, ..., 11
$\|QW\|$	1, 2, 3, **4**, 5
β (% of i-words in QW)	20%, 40%, **60%**, 80%, 100%
δ_{s2t} (meter)	1100, 1300, **1500**, ..., 2100
η	1.4, **1.6**, 1.8, 2.0
α	0.1, 0.3, **0.5**, 0.7, 0.9
τ	0.05, **0.1**, 0.2, 0.4

Efficiency Studies

Performance Overview. We run each algorithm in the default setting and report the running time per query instance in Fig. B.4. Among all, ToE and KoE perform the best because they make full use of all pruning rules. In general, ToE returns top-7 results

[8] http://www.keyworddiscovery.com/keyword-stats.html

within 117ms while KoE needs about 133ms. For ToE\D and KoE\D, the distance-based pruning has a greater impact on their efficiency. Next, ToE\B and KoE\B are basically equal to their original counterparts, showing that the *k*bound pruning barely works in the default setting. The effect of parameter k is studied shortly in the next set of experiments. Still in Fig. B.4, the KoE-based algorithms fluctuate more on different query instances than KoE-based ones. This is because the expansion of KoE is highly related to the query words, and thus is easily influenced by the randomly generated QWs. In contrast, ToE's expansion is relatively stable because it always finds the next door according to indoor topology rather than QWs.

KoE* is much slower than others and it has a wider range of variations. This indicates that its precomputing does not pay off. On the contrary, it needs to recompute indoor distances when a route regularity check fails and the recomputed results cannot be reused in a dynamic routing process. Fig. B.4 omits the results of ToE\P as it is five to six orders of magnitude slower than the others. ToE\P increases the number of routes exponentially due to its absence of prime route-based pruning. As ToE\P and KoE* perform poorly, we omit them in further comparisons but discuss them separately in Sections B.5.1 and B.5.1, respectively.

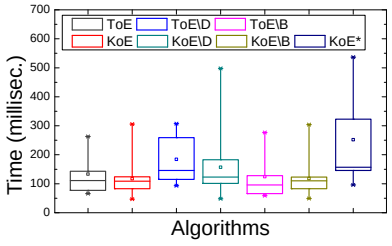


Fig. B.4: Default parameters

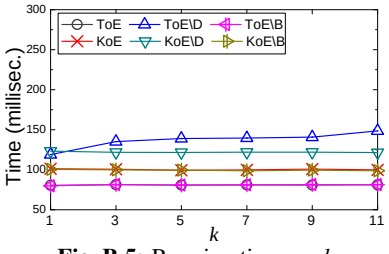


Fig. B.5: Running time vs. k

Effect of k. We investigate the effect of k by varying it from 1 to 11. Referring to Fig. B.5, the running time of each algorithm increases only slightly as k increases. Each KoE variant outperforms its ToE counterpart. Moreover, ToE\D and KoE\D are much slower than ToE and KoE, which again demonstrates the power of the distance-based pruning. Consistent with the default parameter tests, the gap between ToE\B (KoE\B) and ToE (KoE) is insignificant. Sometimes ToE\B is even faster than ToE. When $|QW|$ is at its default of 4, the overestimated keyword relevances of some partial routes tend to be higher than the final keyword relevance of routes already obtained, making the *k*bound less useful to prune those partial routes. Considering the extra *k*bound maintenance costs, ToE can be slower than ToE\B. Nevertheless, both ToE and KoE return the top-11 routes within 150ms.

Effect of $|QW|$. We vary $|QW|$ from 1 to 5 and report the running time and memory costs in Fig. B.6 and B.7, respectively. For all algorithms, both metrics increase when $|QW|$ is larger. Referring to Fig. B.6, all KoE-based algorithms slow down more rapidly than ToE counterparts. When there are more query words, it is more difficult for partial routes to achieve full coverage of query words and connect to the terminal

quickly. Therefore, both ToE and KoE are slower when $|QW|$ increases. Moreover, a larger $|QW|$ leads to more candidate partitions and thus more keyword combinations are considered in KoE. As a result, KoE's running time grows faster than ToE. When $|QW|$ increases to 5, the maximum query keyword size, each KoE-based variant incurs more time than its ToE counterpart. Nevertheless, KoE can still return the top-7 routes within 300ms. Referring to Fig. B.7, KoE family cost less memory than ToE family as KoE expansions are more aggressive, jumping directly from one key partition to another without caching intermediate results, whereas KoE has the lowest memory cost thanks to its efficient route pruning.

Effect of η. Referring to Fig. B.8, when increasing η from 1.6 to 2, both ToE and ToE\B's running time increase steadily since the distance constraint is larger. In contrast, ToE\D is insensitive to η as it does not use any distance-related pruning. On the other hand, KoE family's time costs only slightly increase with η, showing that they can work well with larger or looser distance constraints. Referring to Fig. B.9, when increasing η, the memory costs of ToE family increase while those of KoE family stay stable, which again demonstrates KoE family's insensitiveness to the distance constraint.

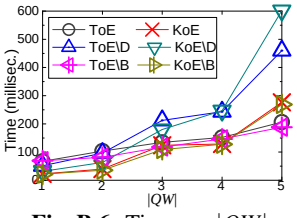


Fig. B.6: Time vs. $|QW|$

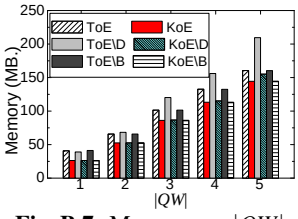


Fig. B.7: Memory vs. $|QW|$

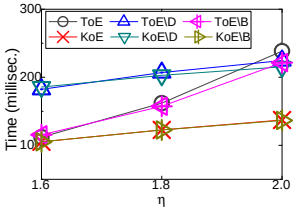


Fig. B.8: Time vs. η

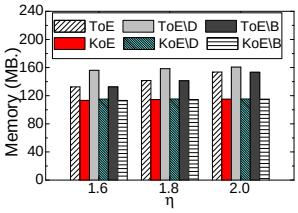


Fig. B.9: Memory vs. η

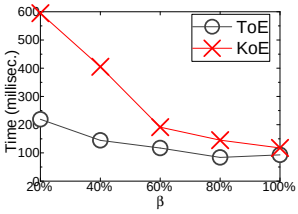


Fig. B.10: Time vs. β

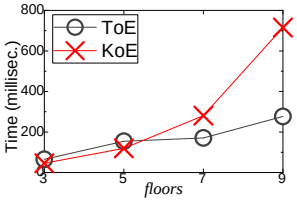


Fig. B.11: Time vs. *floor*

Next, we concentrate on comparing ToE and KoE.

Effect of β. Referring to Fig. B.10, both algorithms speed up clearly when increasing the i-word faction β. As each t-word may relate to more partitions than each i-word in our setting, a larger β tends to exclude more t-words and thus more candidate partitions. Therefore, both algorithms return the results faster for queries with more i-words. Still, ToE outperforms KoE and the gap enlarges rapidly when varying β from 60% to 20%. That is because the candidate i-word set will be large with more t-words, which more affects KoE.

Effect of floor number. We vary the floor number to test the scalability of our algorithms. Referring to Fig. B.11, ToE's time cost increases slowly but KoE deteriorates very fast when there are more floors. The distance between adjacent floors in our dataset is set to 20m only, which means the distance between two points separated by several floors is still very small. Consequently, the distance constraint can hardly help exclude the candidate partitions several floors away. Thus, both search algorithms need to consider more candidates. Nevertheless, ToE can still finish within 250ms when there are 9 floors. As ToE keeps the intermediate results at each step, its running time increases slower than KoE for more floors.

Effect of δ_{s2t}. We vary the route distance δ_{s2t} with η fixed to 1.6. Referring to Fig. B.12, both algorithms slow down sightly with δ_{s2t} increased to 1900m. When δ_{s2t} is small, ToE that expands based on topology can quickly find enough routes and return. However, when p_s and p_t are separated further, ToE needs to expand more partitions and thus costs more time. In contrast, KoE finds the next valid stamp based on keywords and is less affected by the increase of δ_{s2t}.

Effect of α and τ. With varying α, all algorithms perform steadily with minor fluctuations only. This implies that our ranking score is robust and insensitive to α. The experiments with varying τ show that our search algorithms are also insensitive to τ. The Jaccard similarity in our keyword relevance is rather long-tailed. Very few indirect matching i-words are retrieved even τ is tuned to 0.05. Thus our search algorithms stay stable. Due to page limit, we omit the result figures.

Summary. In general, KoE has better scalability when some distance-related parameters (e.g., η and δ_{s2t}) are enlarged. Conversely, ToE is more efficient when there are more query words. In addition, KoE always has a lower memory cost.

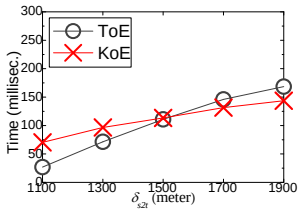


Fig. B.12: Time vs. δ_{s2t}

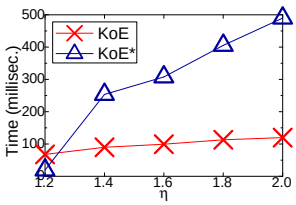


Fig. B.13: Time of KoE*

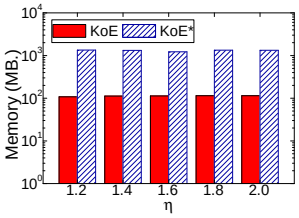


Fig. B.14: Memory of KoE*

Effect of Precomputing in KoE

With others in default, we run KoE and KoE* at different η values. Referring to Fig. B.13, KoE always outperforms KoE* except when η is as small as 1.2. A smaller η leads to a tighter distance constraint, and KoE tends to directly connect to p_t with the shortest distance regardless of covering query words. In such a case, the precomputed shortest routes between key partitions are useful. However, once the distance constraint becomes larger, more routing choices are included and the precomputed results become useless. This leads to a lot of recomputations that clearly jeopardize

KoE*'s efficiency. As shown in Fig. B.14, KoE*'s memory cost is an order of magnitude higher than that of KoE as it uses precomputing. In summary, we find that KoE's on-the-fly search nature yields much more performance gains in both time and memory costs than KoE*'s precomputing.

Effect of Prime Route-based Pruning

We compare ToE to ToE\P that does not employ the prime route-based pruning. Referring to Fig. B.15, when increasing η from 1.4 to 2, ToE\P slows down almost exponentially whereas ToE stays stable. As ToE\P never checks and prunes those non-prime routes during the search, its candidate routes can be extremely large even when a small η is used. When η increases to 2, ToE\P is three orders of magnitude slower than ToE.

Without the prime concept, ToE\P tends to return homogeneous routes. We measure the **homogeneous rate** as the fraction of homogeneous routes in the returned top-k routes. The results w.r.t. different k values are reported in Fig. B.16. With a larger k, ToE\P's top-k routes become homogeneous at a rapid pace. For $k \geq 3$, more than 60% of returned routes are homogenous, and the percentage grows up 92% when k is 15. Such top-k results are barely interesting to users. Since ToE\P also runs fast as shown in Fig. B.15, it is of great importance to perform the prime route-based pruning in our search.

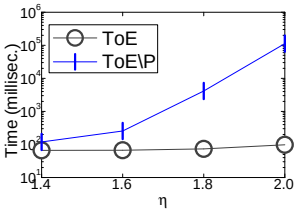


Fig. B.15: Time of ToE\P

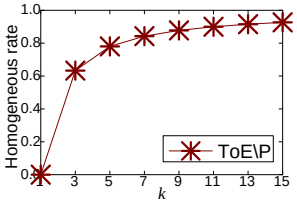


Fig. B.16: Homogeneous rate

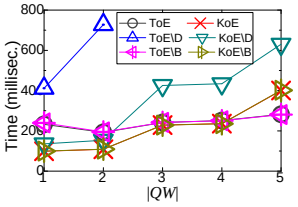


Fig. B.17: Time vs. $|QW|$

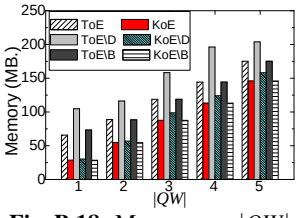


Fig. B.18: Memory vs. $|QW|$

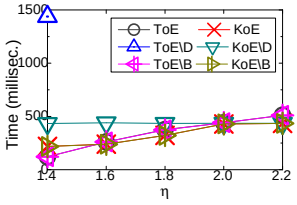


Fig. B.19: Time vs. η

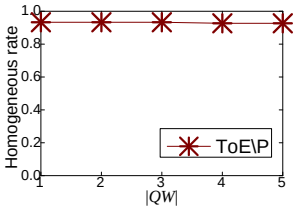


Fig. B.20: Homogeneous rate

Search Result Quality

We use a typical example to show that our IKRQ can find more reasonable and desirable routes in practice. Referring to Fig. B.1, we have $I2T(Apple) = \{phone, mac,$

laptop, *watch*} and $I2T(Samsung) = \{phone, laptop, earphone\}$. Assuming α is 0.5 and τ is 0.1, query $(p_1, p_2, 100, earphone, 2)$ returns routes $R_1 = (p_1, d_4, d_{15}, d_{15}, p_2)$ and $R_2 = (p_1, d_4, d_{17}, d_{17}, p_2)$, although *earphone* is not in Apple's t-words in R_1. In particular, $\delta(R_1) = 10m$, $\delta(R_2) = 20m$, $\rho(R_1) = 1.667$ and $\rho(R_2) = 2$, so $\psi(R_1) = 0.867$ and $\psi(R_2) = 0.9$. Although $R_3 = (p_1, d_4, p_2)$ has a shorter distance of $9.5m$, it lacks words similar with *earphone* and is not returned with $\psi(R_3) = 0.4525$. Apparently, Apple offers earphones. Its route R_1 will be excluded and users will miss useful choices if we use exact keyword matching.

B.5.2 Results on Real Data

We collect a dataset with real indoor topology and keyword distributions from a seven-floor, 2700m $\times$ 2000m shopping mall in Hangzhou, China. There are ten staircases in which each stairway roughly 20m long. Among all the 639 stores, those of the same category, e.g., cosmetics and men's wear, are on the same floor(s). We extract the keywords from the store descriptions on the mall's website and obtain 5036 t-words for 533 i-words (stores). There are 103 stores with no t-words but only one i-word. An i-word corresponds to 31 t-words maximum and 9.4 ones on average. We use the same parameter settings as in Table B.4, except that α is adjusted to 0.7 to suit the needs of keyword-awareness in shopping. Like on the synthetic data, we still generate 10 query instances for each parameter setting, run each instance 5 times, and measure the average cost per run for each query instance.

First, we vary $|QW|$. Referring to Fig. B.17, all algorithms but ToE\D moderately incur more time with increasing $|QW|$. Those without distance-based pruning worsen rapidly, e.g., ToE\D cannot return within 1 second when $|QW|$ exceeds 3. Consistent with the results in synthetic data, KoE worsens faster than ToE as $|QW|$ increases, and it becomes less efficient when $|QW| = 5$. In the real mall, shops of the same category are spatially adjacent, resulting in a dense distribution of the candidate partitions that refer to the same query keyword. When distance constraint is certain, KoE needs to consider more partition combinations that complicate the search. In contrast, ToE always expands based on topology and is less affected. As shown in Fig. B.18, the memory cost of each algorithm increases moderately with a larger $|QW|$. However, KoE is always the most space-efficient one.

Also, we study the effect of η on running time. Referring to Fig. B.19, when η increases, i.e., the distance constraint is looser or larger, ToE family needs to access more doors and thus takes more time to return. With looser distance constraints, KoE gradually approaches KoE\D. In this case, all KoE algorithms tend to cover more query words, and therefore they become similar in processing candidate partitions. In general, ToE and KoE can always return the results less than 500ms, showing they are both efficient in finding routes in real applications.

Fig. B.20 reports ToE\P's homogeneous rate in the real data. Without the use of prime routes, ToE\P always returns homogeneous routes, not to mention its high running time.

B.6 Related Work

Indoor Routing and Path Finding. Goetz and Zipf [15] define a routing graph for indoor environments with obstacles. Lu et al. [1] design an indoor space model that facilitates shortest path finding. To speed up distance-aware indoor path finding, Shao et al. [16] design VIP-tree that enables more aggressive pruning. VIP-tree also supports indoor trip planning based on neighbour expansion [17]. Li et al. [18] construct indoor possible paths based on probabilistic location samples of moving objects and search for the most popular indoor semantic regions using the constructed paths. Costa et al. [19] propose context-aware indoor-outdoor path recommendation that minimizes the outdoor exposure and path distance. Li et al. [20] design vision-based mobile indoor navigation that helps blind and visually impaired people walk indoors. In contrast to our IKRQ, these works do not consider indoor semantic keywords. A recent work [21] studies indoor keyword-aware skyline route query that considers the number of covered keywords and route distances, whereas our IKRQ does not count keywords but use prime routes to exclude routes through the same partitions. Also, unlike work [21], our setting allows a partition to have more than one keyword.

Outdoor Keyword-aware Routing. Given a source s, a destination e, and a category set C, the trip planning query [22] finds the shortest s-to-e path that covers at least one object from each category in C, whereas the optimal sequenced route query [23] finds the shortest path covering all categories in a total order. Partial order is considered elsewhere [4]. The multi-approximate-keyword routing query [5] changes the strict category coverage to an approximate matching using edit distances between a keyword and a location. The geographical route search [3] finds routes whose length is within a threshold and keyword-dependent scores are highest. The keyword-aware optimal routing [2] considers keyword coverage, route score, and travel cost budget. The optimal route search [24] finds one route whose word coverage is maximum within a budget constraint. The clue-based route search [25] supports an order of keywords to cover, and requires that the network distance from one matched keyword to next is within a corresponding user-specified limit. However, all these works fall short for indoor topology considered in our IKRQ queries. Also, none of them distinguishes identity and content words that carry different semantics. Moreover, most works do not consider routing diversity, and works [2, 3, 5, 22, 24] are approximate solutions.

B.7 Conclusion

Given two indoor points s and t, indoor top-k keyword-aware routing query (IKRQ) finds k s-to-t routes that have optimal ranking scores integrating keyword relevance and spatial distance constraint. We propose prime routes to increase result diversity, devise data structures for computing route keyword relevances, and derive pruning rules to reduce search space. Further, we design two IKRQ search algorithms that expand differently in routing. Experiments demonstrate the efficiency of our proposals

and the performance characteristics of them.

For future work, we can use a soft distance constraint to support approximate routing. With indoor mobility data, it is possible to incorporate route popularity into routing. Also, it is useful to consider special entities like lifts in routing.

Acknowledgement

This work was supported by HK-RGC (Nos. 12200817 and 12201018), Independent Research Fund Denmark (No. 8022-00366B), and National Science Foundation of China (No. 61672455). The authors would like to thank Ronghao Ni and Yijie Xie for preprocessing the real dataset.

References

[1] H. Lu, X. Cao, and C. S. Jensen, "A foundation for efficient indoor distance-aware query processing," in *2012 IEEE 28th International Conference on Data Engineering*. IEEE, 2012, pp. 438–449.

[2] X. Cao, L. Chen, G. Cong, and X. Xiao, "Keyword-aware optimal route search," *Proceedings of the VLDB Endowment*, vol. 5, no. 11, 2012.

[3] Y. Kanza, E. Safra, Y. Sagiv, and Y. Doytsher, "Heuristic algorithms for route-search queries over geographical data," in *Proceedings of the 16th ACM SIGSPATIAL international conference on Advances in geographic information systems*, 2008, pp. 1–10.

[4] J. Li, Y. Yang, and N. Mamoulis, "Optimal route queries with arbitrary order constraints," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 5, pp. 1097–1110, 2013.

[5] B. Yao, M. Tang, and F. Li, "Multi-approximate-keyword routing in gis data," in *Proceedings of the 19th ACM SIGSPATIAL international conference on advances in geographic information systems*, 2011, pp. 201–210.

[6] L. Qin, J. X. Yu, and L. Chang, "Diversifying top-k results," *Proceedings of the VLDB Endowment*, vol. 5, no. 11, 2012.

[7] G. J. Fakas, Y. Cai, Z. Cai, and N. Mamoulis, "Thematic ranking of object summaries for keyword search," *Data & Knowledge Engineering*, vol. 113, pp. 1–17, 2018.

[8] G. Cong, C. S. Jensen, and D. Wu, "Efficient retrieval of the top-k most relevant spatial web objects," *Proceedings of the VLDB Endowment*, vol. 2, no. 1, pp. 337–348, 2009.

[9] D. Wu, G. Cong, and C. S. Jensen, "A framework for efficient spatial web object retrieval," *The VLDB Journal*, vol. 21, no. 6, pp. 797–822, 2012.

[10] Z. Li, K. C. Lee, B. Zheng, W.-C. Lee, D. Lee, and X. Wang, "Ir-tree: An efficient index for geographic document search," *IEEE transactions on knowledge and data engineering*, vol. 23, no. 4, pp. 585–599, 2010.

[11] R. Hariharan, B. Hore, C. Li, and S. Mehrotra, "Processing spatial-keyword (sk) queries in geographic information retrieval (gir) systems," in *19th International Conference on Scientific and Statistical Database Management (SSDBM 2007)*. IEEE, 2007, pp. 16–16.

[12] X. Xie, H. Lu, and T. B. Pedersen, "Efficient distance-aware query evaluation on indoor moving objects," in *2013 IEEE 29th International Conference on Data Engineering (ICDE)*. IEEE, 2013, pp. 434–445.

[13] S. Rose, D. Engel, N. Cramer, and W. Cowley, "Automatic keyword extraction from individual documents," *Text mining: applications and theory*, vol. 1, pp. 1–20, 2010.

[14] X. Xiao, Q. Luo, Z. Li, X. Xie, and W.-Y. Ma, "A large-scale study on map search logs," *ACM Transactions on the Web (TWEB)*, vol. 4, no. 3, pp. 1–33, 2010.

[15] M. Goetz and A. Zipf, "Formal definition of a user-adaptive and length-optimal routing graph for complex indoor environments," *Geo-Spatial Information Science*, vol. 14, no. 2, pp. 119–128, 2011.

[16] Z. Shao, M. A. Cheema, D. Taniar, and H. Lu, "Vip-tree: an effective index for indoor spatial queries," *Proceedings of the VLDB Endowment*, vol. 10, no. 4, pp. 325–336, 2016.

[17] Z. Shao, M. A. Cheema, and D. Taniar, "Trip planning queries in indoor venues," *The Computer Journal*, vol. 61, no. 3, pp. 409–426, 2018.

[18] H. Li, H. Lu, L. Shou, G. Chen, and K. Chen, "Finding most popular indoor semantic locations using uncertain mobility data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 11, pp. 2108–2123, 2018.

[19] C. Costa, X. Ge, and P. Chrysanthis, "Caprio: Context-aware path recommendation exploiting indoor and outdoor information," in *2019 20th IEEE International Conference on Mobile Data Management (MDM)*. IEEE, 2019, pp. 431–436.

[20] B. Li, J. P. Munoz, X. Rong, Q. Chen, J. Xiao, Y. Tian, A. Arditi, and M. Yousuf, "Vision-based mobile indoor assistive navigation aid for blind people," *IEEE transactions on mobile computing*, vol. 18, no. 3, pp. 702–714, 2018.

[21] C. Salgado, "Keyword-aware skyline routes search in indoor venues," in *Proceedings of the 9th ACM SIGSPATIAL International Workshop on Indoor Spatial Awareness*, 2018, pp. 25–31.

[22] F. Li, D. Cheng, M. Hadjieleftheriou, G. Kollios, and S.-H. Teng, "On trip planning queries in spatial databases," in *International symposium on spatial and temporal databases*. Springer, 2005, pp. 273–290.

[23] M. Sharifzadeh, M. Kolahdouzan, and C. Shahabi, "The optimal sequenced route query," *The VLDB journal*, vol. 17, no. 4, pp. 765–787, 2008.

[24] Y. Zeng, X. Chen, X. Cao, S. Qin, M. Cavazza, and Y. Xiang, "Optimal route search with the coverage of users' preferences," in *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[25] B. Zheng, H. Su, W. Hua, K. Zheng, X. Zhou, and G. Li, "Efficient clue-based route search on road networks," *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, no. 9, pp. 1846–1859, 2017.

Paper C

Towards Indoor Temporal-variation aware Shortest Path Query

Tiantian Liu, Zijin Feng, Huan Li, Hua Lu, Muhammad Aamir Cheema, Hong Cheng, Jianliang Xu

The paper has been published in the
IEEE Transactions on Knowledge and Data Engineering, 2021.

Abstract

The recent years have witnessed the growing popularity of indoor location-based services (LBS) in practice and research. Among others, indoor shortest path query (ISPQ) is of fundamental importance for indoor LBS. However, existing works on ISPQ ignore indoor temporal variations, e.g., the open and close times associated with entities like doors and rooms. In this paper, we define a new type of query called Indoor Temporal-variation aware Shortest Path Query (ITSPQ). It returns the valid shortest path based on the up-to-date indoor topology at the query time. A set of techniques is designed to answer ITSPQ efficiently. We design a graph structure (IT-Graph) that captures indoor temporal variations. To process ITSPQ using IT-Graph, we design two algorithms that check a door's accessibility synchronously and asynchronously. Furthermore, we propose a novel index structure (IT-Index) that extends the state-of-the-art index significantly by storing dynamic door-to-door distances in a compact distance cube associated with tree nodes. When processing ITSPQ using IT-Index, we make use of the distance cube to avoid time-consuming indoor distance computation on-the-fly. We evaluate the proposed techniques using extensive experiments on synthetic and real data. The results show that our IT-Index based method is the most efficient for processing ITSPQ at a modest cost of index memory consumption.

C.1 Introduction

With the recent advancements in indoor positioning technologies and the increasing availability of digital indoor maps, indoor location-based services are becoming increasingly popular. This trend has enabled a wide variety of applications such as helping people navigate through complex buildings, directing people to safe exits during emergency evacuations, tracking staff and equipment in hospitals, and location-based shopping assistance for customers [1–6].

Shortest distance and shortest path queries are among the most fundamental queries for providing various indoor location-based services. Such queries can facilitate people in need. For example, passengers in an airport would like to find the shortest path from his/her current position to the boarding gate. Shortest distance or shortest path queries can also be applied to indoor robots. For example, in automatic warehouses of Amazon, JD.com, and Alibaba, robots can accomplish operational tasks along the shortest paths, e.g., delivering products from one location to another. To support such real-life applications, indoor shortest distance/path queries have received significant research attention [7–9] in the past few years. Shortest distance/path queries in indoor venues pose unique challenges compared to outdoor space (e.g., road networks) because, in the indoor space, movement is enabled and constrained by unique indoor features such as doors, walls, and staircases. Previous research [9] points out that the outdoor techniques are not effective when extended for indoor venues because they

fail to exploit the unique properties of indoor venues. The basic idea behind the existing techniques to answer indoor queries is to model the indoor space as a graph and optionally precompute and materialize distances between certain pairs of doors to enable efficient query processing. For example, the distance matrix [7] precomputes and stores the distances between all pairs of doors in an indoor venue. The state-of-the-art technique, IP-Tree/VIP-Tree [9], reduces the storage requirement by materializing the distances between some selected pairs of doors instead of all pairs.

A major limitation of the existing techniques [3, 7] is that they assume that the whole indoor venue is accessible for navigation and the indoor topology does not change with time. These assumptions do not hold in many real-world scenarios. For example, it is typically desirable to restrict navigation through certain areas of a building, e.g., private offices and meeting rooms in an office building, security areas in an airport, and storage areas in a shopping mall, etc. Similarly, access to some doors may be restricted at certain times of the day, e.g., doors leading to patient wards may only open during visiting hours or certain doors of a shopping mall may close in the evening restricting access to only the shops that are open till late. Such temporal variations significantly affect the indoor topology, which entails a change in the way people can navigate through the building.

Motivated by the aforementioned factors, in this paper, we propose to study *indoor temporal-variation aware shortest path query* (ITSPQ) which returns a shortest path from a source p_s to a target p_t while disallowing navigation *through* private partitions and ensuring that the doors along the path are open when the user reaches there. Unfortunately, the existing techniques cannot handle such queries because 1) the graphs used to model the indoor space do not consider temporal variations; and 2) the precomputed and materialized door-to-door distances become invalid when one or more doors open or close at certain times. For example, the distance matrix may need to be re-computed (or updated) when some doors open or close. The cost to update the existing indexes in real-time may be prohibitive especially for large indoor venues, e.g., the shopping mall that we use in our experimental study has more than 2,000 doors and it is not feasible to update the distance matrix containing over 4 million door-to-door distance entries.

To address the above challenges, we propose an *indoor temporal-variation graph* (IT-GRAPH) which models the indoor topology, semantic properties of indoor entities (e.g., private partitions), geometric information, and temporal variation information in a composite structure. Furthermore, we propose a hierarchical index called *indoor temporal-variation index* (IT-INDEX) which exploits the unique characteristics of the indoor space to facilitate efficient query processing. Additionally, we use distance cubes for the nodes of the IT-INDEX to materialize temporal-variation aware distances between certain pairs of doors in each node. We design algorithms that exploit IT-GRAPH and IT-INDEX to efficiently answer the indoor temporal-variation aware shortest path queries. Our experimental study on real and synthetic data sets shows that our proposed algorithms are efficient and the size of our proposed indexes increases linearly with the size of indoor venue (in contrast to the distance matrix

which has a quadratic cost to the number of doors in the indoor venue).

Below we summarize the contributions made in this paper.

- To the best of our knowledge, this is the first study on indoor temporal-variation aware shortest path queries (ITSPQ). We formally define ITSPQ and summarize why the existing techniques are not fit for ITSPQ (Section C.2).
- We present IT-GRAPH that effectively captures temporal changes and semantic properties of indoor venues (Section C.3).
- We propose the novel index IT-INDEX to materialize temporal-variation aware distances between door pairs, followed by efficient query algorithms for ITSPQ (Section C.4).
- We conduct extensive experiments on both real and synthetic data (Section C.5). The results demonstrate that IT-INDEX incurs low storage cost and short construction time but enables highly efficient processing of ITSPQ.

In addition, we review the related work in Section C.6 and conclude the paper and discuss future directions in Section C.7.

In contrast to our preliminary work [10], this paper contains substantial extensions. First, it provides a technical discussion on why the state-of-the-art techniques fail to work for ITSPQ (Section C.2.3). Second, it presents more technical details with a concrete example of the IT-GRAPH based approaches (Section C.3.2). Third, it proposes the new index IT-INDEX (Section C.4.1), an efficient index based query processing algorithm (Section C.4.2), and a complexity analysis of all algorithms (Section C.4.3). Fourth, it reports on significantly more extensive experimental studies that use both synthetic and real data to evaluate all proposed techniques in a wide variety of settings (Section C.5).

C.2 Preliminaries

Table C.1 lists the frequently used notations in this paper.

Table C.1: Notations

Symbol	Meaning
v, d, p	partition, door, and point in an indoor space
PRD, PBD	private door, public door
PRP, PBP	private partition, public partition
ATI	active time interval
G_{IT}	indoor temporal-variation graph
$AD_{\sqsupset}(N)$	enterable access doors in a tree node N
$AD_{\sqsubset}(N)$	leavable access doors in a tree node N

C.2.1 Differentiation of Indoor Entities

In this paper, we distinguish two types of indoor partitions. **Private partitions** are occupied for a specific use and not used for routing, e.g., someone's office room or a meeting room. On the contrary, **public partitions** can be used in routing, e.g., hallways and staircases. We treat different partition types as a special kind of temporal variation in that they can be used differently at different times. For example, a private office room is not used as an intermediate partition in routing at normal time but it may be used at emergency time. Accordingly, a door that interconnects two public partitions is a **public door** while a door that connects to one or more private partitions is a **private door**. In this sense, a private door can only be the first or the last door in an indoor path.

Example C.2.1 (Partitions)
Referring to the floorplan of an office building in Fig. C.1, someone at point p_1 can get to point p_2 through doors d_3 and d_{17}, but cannot go through d_6 and d_7 to reach p_2 as v_6 is a private office that cannot be passed. Moreover, d_6 is a private door as it connects to a private office v_6 whereas d_3 is a public door as it connects partitions v_3 and v_{16} that are both public hallways.

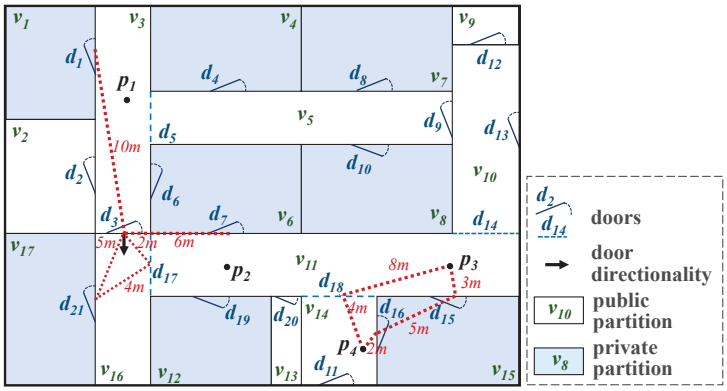


Fig. C.1: An Example of Indoor FloorPlan

For ease of presentation, we show only one floor in Fig. C.1. Nevertheless, our model supports multiple floors where two adjacent floors are connected by a staircase. Specifically, a staircase works as a special partition with two doors—each connects to an adjacent partition at one of the two adjacent floors.

C.2.2 Problem Definition

In real life, we may encounter temporal variations of doors, which can significantly change indoor topology and therefore affect the routing process. For example, the doors in the space illustrated in Figure C.1 may be open and closed at different times as listed in Table C.2. In our setting, we use [open-time, close-time) to represent an **active time interval** (ATI) of a door. Thus, [8:00, 16:00) means a door is open at 8:00 and closed at 16:00. If a door features multiple ATIs, we use an array to store them. Intuitively, the temporal variation of a private door may have little impact on the indoor topology while that of a public door can significantly change the topology.

Table C.2: Active Time Intervals (ATIs) of Doors

Door, **ATIs**	**Door**, **ATIs**
d_1, ⟨[5:00, 23:00)⟩	d_2, ⟨[8:00, 16:00)⟩
d_3, ⟨[6:00, 23:00)⟩	d_4, ⟨[9:00, 18:00)⟩
d_5, ⟨[6:30, 23:00)⟩	d_6, ⟨[8:00, 16:00)⟩
d_7, ⟨[6:00, 23:30)⟩	d_8, ⟨[9:00, 18:00)⟩
d_9, ⟨[0:00, 6:00), [6:30, 23:00)⟩	d_{10}, ⟨[8:00, 16:00)⟩
d_{11}, ⟨[5:00, 23:00)⟩	d_{12}, ⟨[5:00, 23:00)⟩
d_{13}, ⟨[5:00, 17:00), [18:00, 23:00)⟩	d_{14}, ⟨[0:00, 24:00)⟩
d_{15}, ⟨[8:00, 16:00)⟩	d_{16}, ⟨[8:00, 17:00)⟩
d_{17}, ⟨[0:00, 24:00)⟩	d_{18}, ⟨[0:00, 23:00)⟩
d_{19}, ⟨[8:00, 16:00)⟩	d_{20}, ⟨[5:00, 23:00)⟩
d_{21}, ⟨[8:00, 16:00)⟩	

Example C.2.2 (Open Time)
In Table C.2, the door d_1 is open during the time interval [5:00, 23:00). The door d_9 is open at 0:00 and closed at 6:00. It is open again at 6:30 and closed at 23:00. Moreover, closing the private door d_1 only affects those who want to enter or leave the partition v_1. In contrast, closing the public door d_9 will block the direct path between the hallways v_5 and v_{10}, forcing people in nearby partitions to choose alternative paths.

It is noteworthy that the doors may have different ATIs for different days (e.g., weekdays vs weekends). Our techniques can handle such cases by maintaining the date information. Considering the door directionality, a door may also have different ATIs for its two directions. This can be addressed by replacing a door d with two unidirectional doors d_{in} and d_{out} and associating specific ATIs to each of them. To ease the presentation in our setting, we assume that each door features the same ATIs daily, and the ATIs are the same for each door's two directions. Nevertheless, the techniques proposed in this paper can be extended to handle practicalities in real-world scenarios.

On top of the temporal variations of indoor entities, we formulate our research problem as follows.

Problem (Indoor Temporal-variation aware Shortest Path Query). *Given a static start point p_s, a static target point p_t, and a current timestamp t, an* indoor temporal-variation aware shortest path query ITSPQ*(p_s, p_t, t) returns the valid shortest path from p_s to p_t that meets the following rules:*

1. *Each door d_i in the path should be open at $t + \Delta t$[1], where Δt is the walking time from p_s to d_i and it is computed based on human's average walking speed [11] — $5km/h$;*
2. *The path should not go through any private partition except the private partitions that contain p_s and/or p_t.*

Example C.2.3 (Examples of ITSPQ)
Given a query ITSPQ(p_3, p_4, 9:00), we consider two candidate indoor paths, i.e., $(p_3, d_{15}, d_{16}, p_4)$ with length $10m$ and (p_3, d_{18}, p_4) with length $12m$. Although $(p_3, d_{15}, d_{16}, p_4)$ is the shorter one, it goes through a private partition v_{15} that breaks rule 2) in the problem definition. Therefore, the query returns (p_3, d_{18}, p_4) as the result. In contrast, another query ITSPQ(p_3, p_4, 23:30) returns null because d_{18} is closed at 23:00 and no path can meet both rules in the problem definition.

ITSPQ is useful in pertinent indoor applications as it considers the use of indoor space and temporal variations of indoor topology in real life. For example, in an airport or a hospital where rooms fulfill different purposes and doors are dynamically open and closed, a path returned by ITSPQ can help a user quickly reach her destination in the right way at the right time.

C.2.3 Indoor Shortest Distance/Path Query Techniques

Indoor distance computation and path query have been studied in the literature [7, 9].
Indoor Distance-Aware Model [7] considers both geometric and topological information of indoor space as a directed graph (V, E_a, L, f_{dv}, f_{d2d}). Specifically, V is a set of partitions represented as a vertex set, E_a is a set of directed edges, L is doors as edge labels, f_{dv} is a function to compute the maximum distance from a door to all positions within a partition, and f_{d2d} is a function to compute the door-to-door distance. In addition, a distance matrix stores the shortest distances between each door pair. It speeds up the shortest path queries at the costs of extra storage and precomputing.

[1] In this paper, we do not consider the waiting tolerance in the routing, i.e., someone reaches a door and waits there until the door opens.

VIP-Tree [9] is an improved model for indoor shortest distance/path queries. In a VIP-tree, each leaf node consists of a number of 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. Access doors and a distance matrix are maintained in each node. The access door of a node N is a border door which can connect N to the space outside of N. The distance matrix for a leaf node stores the shortest distance (and 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. Given a shortest path query from p_s to p_t, VIP-tree finds the lowest common ancestor of the leaf nodes $\texttt{Leaf}(p_s)$ and $\texttt{Leaf}(p_t)$ that connects the shortest paths from p_s to p_t by access doors. Since only local shortest paths and relevant access doors are maintained at each node, VIP-tree has lower preprocessing costs than the indoor distance-aware model [7].

However, unlike this work, neither the indoor distance-aware model nor VIP-Tree supports temporal variations on doors and different types of partitions. Consequently, the two approaches' materialized shortest distance/path information becomes invalid for ITSPQ and the two approaches fall short in processing ITSPQ. Next, we introduce the indoor temporal-variation graph that can facilitate ITSPQ.

C.3 ITSPQ using Temporal-Variation Graph

We present the structure of the **indoor temporal-variation graph** (IT-GRAPH) in Section C.3.1, and the query processing algorithms based on IT-GRAPH in Section C.3.2.

C.3.1 Indoor Temporal-Variation Graph

To integrate the temporal variations of doors into the indoor topology, we design IT-GRAPH G_{IT} (V, E, L_v, L_E) where

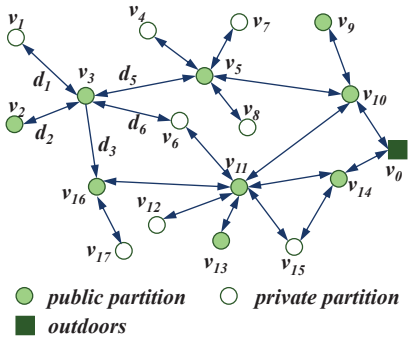
1. V is the set of vertices. Each vertex $v \in V$ is an indoor partition.
2. E is the set of directed edges. Each edge $(v_i, v_j, d_k) \in E$ means one can reach v_j from v_i through a door d_k. We use $\pi_D(E)$ to denote the set of doors associated with the edges of E.
3. L_V is the set of vertex labels. Each vertex label is a 3-tuple(ID_v, *p-type*, *DM*) where ID_v identifies the partition in the vertex, *p-type* = $\{PBP, PRP\}$ indicates if the partition is a public partition (*PBP*) or a private partition (*PRP*), and *DM* is a distance matrix that stores the intra-partition distance between each pair of doors of that partition. *DM* is set to null if the partition has only one door.
4. L_E is the set of edge labels. Each edge label is a 3-tuple (ID_d, *d-type*, *ATIs*) where ID_d identifies the door on the edge, *d-type* = $\{PBD, PRD\}$ indicates if

the door is a public door (*PBD*) or a private door (*PRD*), and *ATIs* are the ATIs (see Section C.2.2) of the door.

The IT-GRAPH corresponding to Fig. C.1 is depicted in Fig. C.2. The partitions are represented by circular vertices. The solid and hollow ones are public and private partitions, respectively. We use a square vertex to denote the outdoor space. The arrows of an edge represent the directionality of the corresponding door. We use a door table and a partition table to store L_V and L_E in IT-GRAPH, respectively. Referring to the tables in Fig. C.2, a record (d_1, *PRD*, $\langle$[5:00, 23:00)$\rangle$) means d_1 is a private door and is open from 5:00 to 23:00. Also, we know v_{16} is a public partition and the distance between its doors d_3 and d_{17} is 2m.

In general, IT-GRAPH combines indoor topology (i.e., graph structure), semantic properties of indoor entities (i.e., *d-type* and *p-type*), geometric information (i.e., *DM*), and temporal variation information (i.e., *ATIs*) in a composite structure.

Following the previous work [7], we also use several mapping functions to facilitate searching between partitions and doors. Specifically, $P2D(v_k)$ maps a partition v_k to the set of doors connected to v_k and $D2P(d_i)$ maps a door d_i to the pair of partitions connected by d_i. Considering the door directionality, $P2D_{\sqsupset}(v_k)$ gives the set of *enterable* doors through which one can enter partition v_k, $P2D_{\sqsubset}(v_k)$ gives the set of *leavable* doors through which one can leave partition v_k, $D2P_{\sqsupset}(d_i)$ 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. Those mappings can be easily obtained based on the connectivity information in IT-GRAPH. Referring to Fig. C.2, we have $D2P(d_3) = \{v_3, v_{16}\}$, $D2P_{\sqsubset}(d_3) = v_3$, and $D2P_{\sqsupset}(d_3) = v_{16}$. Also, we have $P2D(v_3) = P2D_{\sqsubset}(v_3) = \{d_1, d_2, d_3, d_5, d_6\}$ whereas $P2D_{\sqsupset}(v_3) = \{d_1, d_2, d_5, d_6\}$.



Door Table

ID_d	*d-type*	*ATIs*
d_7	PRD	$\langle$[6:00, 23:30)$\rangle$
d_3	PBD	$\langle$[6:00, 23:00)$\rangle$
...	...	...

Partition Table

ID_v	*p-type*	*DM*
v_1	PRP	[[(d_1, d_1), 0]]
v_{16}	PBP	[[(d_3, d_{17}), 2], [(d_3, d_{21}), 4], [(d_{17}, d_{21}), 5]]
...	...	...

Fig. C.2: Example of Indoor Temporal-Variation Graph

C.3.2 IT-GRAPH based ITSPQ Processing

The overall framework for processing ITSPQ based on IT-GRAPH is presented in Algorithm 7.

Algorithm 7 `ITSPQ_ITGraph`(p_s, p_t, t, G_{IT})

Require: Start point p_s, taget point p_t, query time t, and IT-GRAPH G_{IT}
Ensure: A valid shortest path from p_s to p_t at t

1: initialize a min-heap H
2: **for** each door $d_i \in \pi_D(G_{IT}.E)$ **do**
3: $dist[d_i] \leftarrow \infty$
4: enheap(H, $\langle d_i, dist[d_i] \rangle$)
5: $prev[d_i] \leftarrow null$
6: $dist[p_s] \leftarrow 0$; enheap(H, $\langle p_s, dist[p_s] \rangle$)
7: $dist[p_t] \leftarrow \infty$; enheap($H$, $\langle p_t, dist[p_t] \rangle$)
8: **while** H is not empty **do**
9: $\langle d_i, dist[d_i] \rangle \leftarrow$ deheap(H)
10: **if** $dist[d_i] = \infty$ **then return** no such routes
11: **if** $d_i = p_t$ **then**
12: $path \leftarrow p_t$
13: **while** $prev[d_i] \neq p_s$ **do**
14: $path \leftarrow prev[d_i] + ", " + path$
15: $d_i \leftarrow prev[d_i]$
16: $path \leftarrow p_s + ", " + path$
17: **return** $path$
18: **if** $d_i = p_s$ **then** $v \leftarrow \mathsf{P}(p_s)$ **else** $v \leftarrow D2P_{\sqsupset}(d_i) \setminus$ visited partitions
19: mark d_i and v as visited
20: **if** $d_i \in P2D_{\sqsupset}(\mathsf{P}(p_t))$ **then**
21: **if** $dist[d_i] + |d_i, p_t|_E < dist[p_t]$ **then**
22: $dist[p_t] \leftarrow dist[d_i] + |d_i, p_t|_E$
23: enheap(H, $\langle p_t, dist[p_t] \rangle$)
24: $prev[p_t] \leftarrow (v, d_i)$
25: **else**
26: **for** each unvisited door $d_j \in P2D_{\sqsubset}(v)$ **do**
27: $v' \leftarrow D2P_{\sqsupset}(d_j) \setminus v$
28: **if** $v'.d\text{-}type$ is PRP **then** continue
29: $dist_j \leftarrow dist[d_i] + DM(v, d_i, d_j)$; $t_c \leftarrow t + dist[d_i] / velocity$
30: **if** !`TV_Check`$(d_j, DM(v, d_i, d_j), t_c)$ **then** continue
31: **if** $dist_j < dist[d_j]$ **then**
32: $dist[d_j] \leftarrow dist_j$
33: enheap(H, $\langle d_j, dist[d_j] \rangle$)
34: $prev[d_j] \leftarrow (v, d_i)$

The algorithm first initializes a min-heap H to keep the pairs of a door and the distance from p_s to this door (line 1). The min-heap is prioritized according to the distance. The framework then goes through each door d_i in G_{IT} (line 2), initializes $dist[d_i]$ that is the current shortest distance from p_s to d_i (line 3), and enheaps all of them into H (line 4). Besides, $prev[d_i]$ keeps the last hop door of the shortest path from p_s to d_i and is initialized to null for each door d_i (line 5). The algorithm also initializes the shortest distance information for p_s and p_t, and enheaps them into H (lines 6–7). It then iterates on H to search for the shortest path from p_s to p_t (lines 8–34). First, it deheaps a door (or a point) d_i with the minimum distance $dist[d_i]$ (line 9). If $dist[d_i]$ is ∞, meaning all remaining unvisited doors cannot get to p_t, "no such routes" is returned (line 10). If d_i equals p_t, the shortest *path* will be returned by iteratively concatenating the last hops from $prev[d_i]$ (lines 11–17). Otherwise, the framework searches the next partition v for the current d_i. In particular, if d_i equals p_s, v is p_s's covering partition $\mathsf{P}(p_s)$. If not, v is obtained as the enterable partition of d_i that has not been visited (line 18). After that, d_i and v are marked as visited (line 19).

Next, if d_i is an enterable door of p_t's covering partition $\mathsf{P}(p_t)$ (line 20), the next hop of the shortest path should be p_t. In this case, the framework directly updates $dist[p_t]$ and $prev[p_t]$ if $dist[p_t]$ is smaller than the current shortest path distance in $dist[p_t]$ (lines 21–24). Otherwise, the framework tests each unvisited door d_j in v's leavable door set (lines 25–34). In particular, the next partition v' after d_j is obtained (line 27) and d_j is immediately discarded if v' is private (line 28). Then, the current path distance $dist_j$ from p_s to d_j is obtained as the sum of $dist[d_i]$ and the distance from d_i to d_j through v, and the current time t_c is obtained as query time t plus the time cost from p_s to d_i (line 29). Next, the framework calls a function `TV_Check`$(d_j, DM(v,d_i,d_j), t_c)$ to check if d_j is open at the arrival time relative to the current time t_c (line 30). Two different strategies, namely `Syn_Check()` (Algorithm 8) and `Asyn_Check()` (Algorithm 10) are used for this function. Their details are to be given below. Afterwards, the shortest distance and last hop information of the validated door d_j is updated if the current path distance $dist_j$ is smaller than d_j's best one so far (lines 31–34).

Example C.3.1 (An Example of IT-GRAPH based ITSPQ Processing)
Corresponding to the ATIs information in Table C.2, we want to find the shortest path from p_1 to p_2 at 11:00 in Fig. C.1. To this end, we first find all the doors through which one can leave v_3 (the host partition of p_1), i.e., d_1, d_2, d_3, d_5, and d_6. As d_1 and d_6 connect to private partitions that are not p_2's host partition, they are filtered out. Then, we compute the distances from the current node p_1 to the remaining doors d_2, d_3, and d_5, and push each door and its distance from p_1 to a min-heap. Next, the nearest door from p_1 (i.e., d_5) is deheaped as the new current node. Such an expansion to the next node is repeated until a deheaped door is an enterable door of p_2's host partition. By connecting the last node to p_2, we obtain a satisfactory path (p_1, d_3, d_{17}, p_2).

Synchronous Check. The idea of is to look up a door d's *ATIs* and compare it to the arrival time when one just leaves for d. In Algorithm 8, the arrival time t_{arr} is computed as the current time t_c plus the travel time ($dist/velocity$) to go through the distance $dist$ from the previous door to d (line 1). The function returns false if t_{arr} is not in the door d's ATIs, and true otherwise.

Algorithm 8 `Syn_Check`(d, $dist$, t_c)

Require: A door d, the distance $dist$, and current time t_c
Ensure: A result whether the door is valid
1: $t_{arr} \leftarrow t_c + dist/velocity$
2: **if** $t_{arr} \notin d.ATIs$ **then return** *false* **else return** *true*

Asynchronous Check. The synchronous check needs to validate each encountered door by comparing the arrival time with the door's active time intervals. However, in usual scenarios, the temporal variation of doors in IT-GRAPH can only happen at several particular open or close times. We call such time points *checkpoints*. The topology information will *not* change between two consecutive checkpoints. For example, in Table C.2 we can find a set T of the checkpoints as (0:00, 5:00, 6:00, 6:30, 8:00, 9:00, 16:00, 17:00, 18:00, 23:00, 23:30, 24:00). The topology between 9:00 to 16:00 remains the same as depicted in the left of Fig. C.2. In contrast, when time goes between 16:00 and 17:00, the topology will be changed to the one illustrated in Fig. C.3. A red cross on an edge means the corresponding door is closed between 16:00 and 17:00. As such, an alternative checking strategy is to directly refer to a

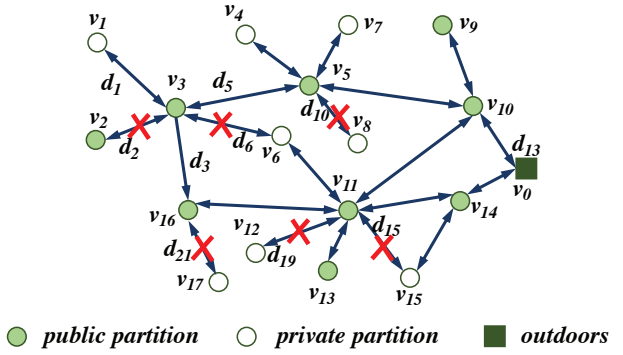


Fig. C.3: Indoor Temporal-Variation Graph within [16:00,17:00)

time-dependent IT-GRAPH that only keeps all currently open doors. The information of IT-GRAPH only needs to be updated asynchronously at the next checkpoint. Given the set T of checkpoints, model updating procedure at a time t_{arr} is presented in Algorithm 9. First, it initializes a new graph G'_{IT} using the initial graph G^0_{IT} that keeps the original indoor topology without considering temporal variations. Next, it searches the previous checkpoint cp relative to t_{arr} (line 2), and obtains the set D_c of doors

that *have been* closed at *cp* (line 3). Afterwards, it goes through each such door d_i in D_c and removes its every relevant edge $(\cdot,\cdot,d_i)$ in G'_{IT} (lines 4–5). Note that we only need to remove the closed doors at checkpoint *cp* from the complete topology G'_{IT}, and it has nothing to do with graph instances at other time points. Finally, it returns *cp* along with the new model G'_{IT} (line 6). G'_{IT} takes effect in the further iteractions of Algorithm 7.

Algorithm 9 `Graph_Update`(t_{arr}, T)

Require: Arrival time t_{arr} and checkpoints set T
Ensure: An updated graph G'_{IT} along with t_{arr}'s previous checkpoint *cp*
1: $G'_{IT} \leftarrow G^0_{IT}$
2: $cp \leftarrow$ `Find_Previous_Checkpoint`(t_{arr}, T)
3: $D_c \leftarrow$ `Get_Closed_Door`(cp)
4: **for** each door $d_i \in D_c$ **do**
5: remove all edges $(\cdot,\cdot,d_i)$ from $G'_{IT}.E$
6: **return** (cp, G'_{IT})

Based on the graph updating in Algorithm 9, we present the asynchronous check in Algorithm 10. It first gets the current G_{IT} and its corresponding checkpoint *cp* (see line 6 in Algorithm 9) and the arrival time t_{arr} (lines 1–2). Next, if t_{arr} to get to d is later than the next checkpoint in T, it updates G_{IT} using G'_{IT} returned by Algorithm 9 (lines 3–5). Here, we directly update the graph to the latest checkpoint to t_{arr} because the object will not leave the current partition during $[t_c, t_{arr})$ (see line 2). In other words, any topology changes within $[t_c, t_{arr})$ make no difference to the routing. A *true* is returned to keep consistent with the interface of Algorithm 8 (line 6). It ensures that the expansion in lines 31-34 of Algorithm 7 will be executed.

Algorithm 10 `Asyn_Check`(d, $dist$, t_c)

Require: A door d, the distance $dist$, and current time t_c
Ensure: A result whether the door is valid
1: get the current G_{IT} and its corresponding *cp* for time t_c
2: $t_{arr} \leftarrow t_c + dist/velocity$
3: **if** $t_{arr} >$ `Find_Next_Checkpoint`(cp, T) **then**
4: $(cp^*, G'_{IT}) \leftarrow$ `Graph_Update`(t_{arr}, T)
5: $(cp, G_{IT}) \leftarrow (cp^*, G'_{IT})$
6: **return** *true*

Compared to the search using the synchronous check, the search using the asynchronous check involves reduced versions of IT-GRAPH in the outward expansion (lines 18–34 in Algorithm 7), thus pruning some impossibly opening doors in advance and reducing the costs of checking temporal variations.

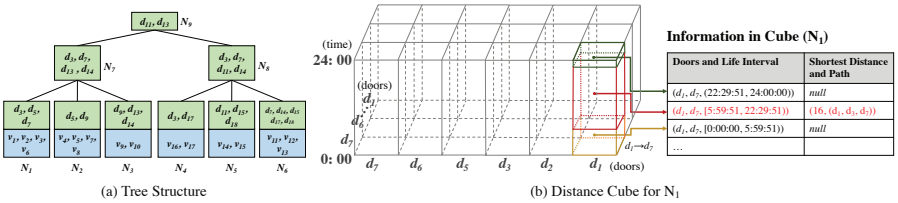
In general, the two searches are suitable for different scenarios. The search using the synchronous check can deal with improvised variations, e.g., when a fire happens

in a building and some doors close urgently. The search using the asynchronous is more suitable for the scenario where doors are opened and closed at fixed time points. In this case, an asynchronous check saves more search costs without on-the-fly handling of ATIs. These two searches are experimentally compared in Section C.5.1.

C.4 ITSPQ using Temporal-Variation Index

In Section C.4.1, we present the **indoor temporal-variation index** (IT-INDEX) that organizes indoor partitions into a tree structure based on indoor topology. The indoor topology here refers to physical layout only and does not involve temporal variations and directionality of doors. Subsequently, we present a query processing algorithm based on IT-INDEX in Section C.4.2. Finally, we analyze the complexity for all ITSPQ approaches in Section C.4.3.

C.4.1 Indoor Temporal-Variation Index



(a) Tree Structure (b) Distance Cube for N_1

Fig. C.4: Indoor Temporal-Variation Index

Considering indoor topology, we find that a valid shortest path should never go through public partitions with only one door (except the partitions that contain p_s and/or p_t). Therefore, we further differentiate partitions into two types for indexing use. In particular, **impassable partitions** include all private partitions and those public partitions with only one door. In contrast, **passable partitions** are public partitions with two or more doors. Referring to Fig. C.1, the private partition v_1 and the one-door public partition v_2 are both impassable partitions, whereas v_3 and v_{11} are passable partitions.

We proceed to present the structure of IT-INDEX. In particular, a set of topologically interconnected partitions form a **leaf node**, and a set of interconnected leaf nodes further form a non-leaf node. The non-leaf nodes are hierarchically merged to form a non-leaf node at a higher level until one root node at the highest level is formed. Corresponding to Fig. C.1, the tree structure of IT-INDEX is illustrated in Fig. C.4(a).

In IT-INDEX, each leaf node N_i maintains a set of *access doors* [9]. Based on door directionality, we distinguish **enterable access doors** and **leavable access doors** for N_i, the doors through which one can enter and leave N_i, respectively. A non-leaf node maintains the pointers for its access doors. It can be shown that the children of

a non-leaf node are interconnected by the access doors of its children. We omit the proof due to the page limit.

The tree construction of IT-INDEX follows the same overall procedure of IP-tree [9]. However, as a special rule, each leaf node in IT-INDEX must contain at least one passable partition. This rule guarantees that any partition in a leaf node can be physically reached (without considering temporal variations and door directionality) via a passable partition that connects to it in that node. Moreover, as a leaf node maintains the shortest distance information for each pair of doors in it, the shortest distance computation will be complicated if a leaf node contains too many passable partitions with multiple public doors. Therefore, we set another rule that only one of the passable partitions in a leaf node can have more than k public doors. We chose $k = 4$ in our implementation according to the evaluation in previous work [9].

Example C.4.1 (An Example of IT-INDEX)
Referring to the tree structure in Fig. C.4(a) that corresponds to the example in Fig. C.1, four interconnected partitions v_1, v_2, v_3, v_6 form a leaf node N_1 and the access doors of N_1 are d_3, d_5, d_7. Particularly, N_1 connects to another leaf node N_2 via the access door d_5. Note that d_5 is also an access door for N_2. Moreover, d_3 is a leavable access door for N_1 due to its door directionality. Three interconnected leaf nodes N_1, N_2, and N_3 form a non-leaf node N_7. N_7's access doors to its outside, i.e., d_3, d_7, d_{13}, d_{14}, are stored.

Each tree node also maintains a three-dimensional structure called *distance cube* to store the shortest path information relevant to that node. One dimension of the cube refers to time and the other two refer to doors. Specifically, we use a 3-tuple (d_i, d_j, L) to denote the index of a distance cube, where d_i, d_j are two doors, and L is a life interval during which the shortest path is valid (considering temporal variation). The value in each cell indexed by (d_i, d_j, L) is denoted as ($dist$, ϕ), where $\phi = (d_i, \ldots, d_j)$ is the shortest path from d_i to d_j within the life interval L and $dist$ is corresponding path distance. The shortest path here conforms to our rule that one can not pass any private door. An example of distance cube for N_i in the tree is depicted in Fig. C.4(b). Given the door pair of d_1 and d_7, the shortest path information is divided into three parts due to the temporal variation of doors. For example, the second record in the table indicates that the shortest path from d_1 to d_7 during the life interval [5:59:51, 22:29:51) is (d_1, d_3, d_7) and its path length is 16m. Compared to IP-tree [9], IT-INDEX maintains semantic properties and temporal variations of indoor entities, together with the distance cube that keeps the shortest distance information with respect to temporal variations. Next, we detail the construction of distance cube.

Construction of Distance Cube. Algorithm 11 constructs the distance cube DC (initialized in line 1) for a leaf node N_i by calling a function `Cell_Build` to compute the shortest path for each pair of an access door and a door in N_i (lines 2–4). Note that we do not need to keep the shortest path for a pair of non-access doors. If N_i is a

Algorithm 11 `Cube_Build`(N_i)

Require: A node N_i
Ensure: The distance cube for node N_i

initialize cube $\mathsf{DC} : (door \times door \times [t_s, t_e]) \rightarrow (dist, \phi)$
for each access door $d_i \in N_i$ **do**
  for each door $d_k \in N_i$ **do**
    `Cell_Build`(d_i, d_k, DC); `Cell_Build`(d_k, d_i, DC)
function `Cell_Build`(d_s, d_t, DC)
  $t_1 \leftarrow 0$; $t_2 \leftarrow 0$
  while $t_2 < 24{:}00$ **do**
    initialize a min-heap H; initialize set $R \leftarrow \varnothing$
    for each door $d_i \in \pi_D(G_{IT}.E)$ **do**
      if $d_i \neq d_s$ **then**
        $d_i.dist \leftarrow \infty$; $d_i.t_{arr} \leftarrow \infty$, $d_i.t_l \leftarrow \infty$
      else
        $d_i.dist \leftarrow 0$; $d_i.t_{arr} \leftarrow t_1$; $d_i.t_l \leftarrow t_1$
      $enheap(H, \langle d_i, d_i.dist \rangle)$
    while H is not empty **do**
      $\langle d_i, d_i.dist \rangle \leftarrow$ deheap(H)
      if $d_i.dist = \infty$ **then**
        $t_2 \leftarrow min(R)$; $\mathsf{DC}[d_s, d_t, [t_1, t_2)] \leftarrow (null, \infty)$
        break
      if $d_i = d_t$ **then**
        $\phi \leftarrow$ concatenate shortest paths from d_s to d_i
        $t_2 \leftarrow d_i.t_l$
        $\mathsf{DC}[d_s, d_t, [t_1, t_2)] \leftarrow (d_i.dist, \phi)$
        break
      mark d_i as visited
      if $d_i.t_{arr} \notin d_i.ATIs$ **then**
        $t_o \leftarrow$ `Get_Next_Open_Time`$(d_i, d_i.t_{arr})$
        $R.add(t_o - d_i.dist/velocity)$
        continue
      for each partition $v \in D2P_{\sqsupset}(d_i)$ **do**
        if $d_i \neq d_t$ and $v.d{-}type$ is PRD **then continue**
        for each unvisited door $d_j \in P2D_{\sqsubset}(v)$ **do**
          if $d_i.dist + DM(v, d_i, d_j) < d_j.dist$ **then**
            $d_j.dist \leftarrow d_i.dist + DM(v, d_i, d_j)$
            $d_j.t_{arr} \leftarrow DM(v, d_i, d_j)/velocity + d_i.t_{arr}$
            if $d_j.t_{arr} \notin d_j.ATIs$ **then** $d_j.t_l \leftarrow d_i.t_l$
            else
              $t_c \leftarrow$ `Get_Next_Close_Time`$(d_j, d_j.t_{arr})$
              $d_j.t_l \leftarrow min(t_c - d_j.dist/velocity, d_i.t_l)$
            $enheap(H, \langle d_j, d_j.dist \rangle)$
    $t_1 \leftarrow t_2$

non-leaf node, the shortest path is computed for each pair of access doors of N_i's child nodes instead.

Function `Cell_Build` (lines 5–41) updates DC for a door pair (d_s, d_t) by going through the time range using two variables t_1 and t_2 (lines 6–7 and 41). In particular, a min-heap H is initialized by inserting each door d_i in IT-GRAPH (line 8), and each d_i is associated with a shortest distance $d_i.dist$ from d_s to d_i, an arrival time t_{arr} to get to d_i, and a latest departure time t_l from d_s at which one can get to d_i (lines 9–13). H is prioritized according to $d_i.dist$ (line 14). A set R is also initialized to keep the next earliest open timestamps of close doors. The function then iterates through each d_i deheaped from H. If d_i's shortest distance from d_s is infinity, the end time t_2 is obtained as the current minimum timestamp in R, and the path from d_s to p_t during the life interval $[t_1, t_2)$ is set to null (lines 17–19). If d_i equals p_t, the function constructs the shortest path ϕ in the same way as does the counterpart (see lines 12–16) of Algorithm 7, setting t_2 as the lastest departure time of d_i, and updates DC with ϕ in the life interval $[t_1, t_2)$ (lines 20-24). If d_i is closed when a path reaches it (line 26), the function obtains the next open time t_o of d_i, computes the earliest time at which one can reach d_i from d_s, i.e., $t_o - d_i.dist/velocity$, and adds it to the set R (lines 26–29). Otherwise, the function goes through each enterable partition v of d_i and finds the next unvisited door d_j (lines 30–32). The latest departure time t_l of d_j is updated in two different cases: If d_j is closed when a path gets to it, t_l of d_j should be the same as its previous-hop door d_i (line 36). Otherwise, t_l of d_j is obtained as the earlier one between the latest time to get to d_j before d_j closes and the lastest departure time of d_i (lines 37–39).

C.4.2 IT-INDEX based ITSPQ Processing

Fig. C.5 illustrates the three different methods introduced in this paper, including the aforementioned two methods based on IT-GRAPH (i.e., ITG/S using synchronous check and ITG/A using asynchronous check) and ITI to be detailed in this section.

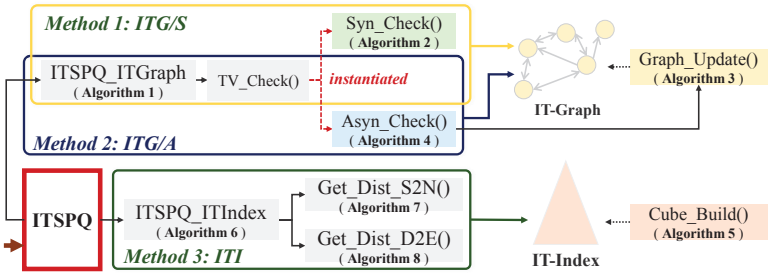


Fig. C.5: Different Methods for ITSPQ Processing

The overall framework of ITI is given in Algorithm 12. It first searches IT-INDEX $Index_{IT}$ for the *lowest common ancestor* N_{LCA} for the points p_s and p_t (line 1). Next,

it obtains two children of N_{LCA}, namely N_s the ancestor of $\texttt{Leaf}(p_s)$ and N_t the ancestor of $\texttt{Leaf}(p_t)$ (lines 2–3). Two variables $dist^*$ and $path^*$ are initialized to keep the shortest distance and path found so far (line 4). It then calls a function `Get_Dist_S2N` to compute the shortest paths from p_s to each leavable access door of N_s (line 5). Afterwards, it iterates through each such leavable access door d_i and checks its temporal variation and semantic properties (line 8). For each qualified d_i, it computes the shortest distance from d_i to each enterable access door of N_t, using the distance cube maintained at N_{LCA} (line 9–10). For each qualified d_j (line 11), it computes the shortest distance from d_j to p_t by calling a function `Get_Dist_D2E` (line 9–10). Consequently, the shortest distance $dist$ from p_s to p_t through d_i and d_j is computed as the sum of the shortest distances of $p_s \rightarrow d_i$, $d_i \rightarrow d_j$ and $d_j \rightarrow p_t$ (line 13). It updates the shortest path $path^*$ if the current distance $dist$ is smaller than $dist^*$ (lines 14–17). Finally, it returns the $path^*$ when the loop is complete. Next, we detail functions `Get_Dist_S2N` and `Get_Dist_D2E`, respectively.

Algorithm 12 `ITSPQ_Index`(p_s, p_t, t, $Index_{IT}$)

Require: Start point p_s, target point p_t, query time t, and $Index_{IT}$
Ensure: The valid shortest path from p_s to p_t at t

1: $N_{LCA} \leftarrow LCA(\texttt{Leaf}(p_s), \texttt{Leaf}(p_t))$ in $Index_{IT}$
2: $N_s \leftarrow$ children of $N_{LCA} \cap$ ancestors of $\texttt{Leaf}(p_s)$
3: $N_t \leftarrow$ children of $N_{LCA} \cap$ ancestors of $\texttt{Leaf}(p_t)$
4: $dist^* \leftarrow \infty$; $path^* \leftarrow null$
5: $S2N \leftarrow \texttt{Get\_Dist\_S2N}(p_s, N_s, t)$
6: **for** $d_i \in AD_{\sqsubset}(N_s)$ **do**
7: $\quad \Delta t_1 \leftarrow S2N[d_i] / velocity + t$
8: $\quad$ **if** $\Delta t_1 \notin d_i.ATIs$ or $d_i.d\text{-}type$ is PRD **then continue**
9: $\quad$ **for** $d_j \in AD_{\sqsupset}(N_t)$ **do**
10: $\quad\quad \Delta t_2 \leftarrow N_{LCA}.\mathsf{DC}[d_i, d_j, \Delta t_1] / velocity + \Delta t_1$
11: $\quad\quad$ **if** $\Delta t_2 \notin d_j.ATIs$ or $d_j.d\text{-}type$ is PRD **then continue**
12: $\quad\quad d2e \leftarrow \texttt{Get\_Dist\_D2E}(d_j, p_t, t + \Delta t_2)$
13: $\quad\quad dist \leftarrow S2N[d_i] + N_{LCA}.\mathsf{DC}[d_i, d_j, \Delta t_1] + d2e$
14: $\quad\quad$ **if** $dist < dist^*$ **then**
15: $\quad\quad\quad dist^* \leftarrow dist$
16: $\quad\quad\quad path^* \leftarrow$ concatenate shortest paths of
17: $\quad\quad\quad\quad p_s \rightarrow d_i$, $d_i \rightarrow d_j$ and $d_j \rightarrow p_t$
18: **return** $path^*$

`Get_Dist_S2N` (Algorithm 13) returns an array $S2N$ (initialized in line 1) that keeps the shortest distance from a point s to each leavable access door of N_s at a current time t_c. The algorithm finds the shortest distance from s to N_s towards the root node by using variables N_c and PN_c (lines 2–3 and 12). Here, N_c is the current node in process and PN_c is N_c's parent node to be processed next. For each N_c, we obtain a candidate leavable access door set $CAD_{\sqsubset}(N_c)$ by removing those unqualified doors

in its leavable access door set $AD_{\sqsubset}(N_c)$. If the time at which one reaches a leavable access door d_i is not in the *ATIs* of d_i, d_i should be removed (line 7). Here, $\texttt{TSD}(s,d,t_c)$ is a *time-dependent shortest distance function* that returns the shortest distance from a point *pt* (either the point s or an access door) to an access door d with respect to current time t_c. Its details are to be given shortly. If d_i is a private door and is not the first door when one leaves the partition containing s, d_i should also be removed (line 8). Afterwards, the algorithm iterates over each leavable access door d in the parent node PN_c (lines 9–11). It marks d as processed and computes its shortest distance from s. If d is a door in $CAD_{\sqsubset}(N_c)$, it records the shortest distance in *S2N* accordingly.

Algorithm 13 `Get_Dist_S2N`(s, N_s, t_c)

Require: A point s, a node N_s, and current time t_c
Ensure: The shortest distance from s to N_s at t_c

```
1:  initialize an array S2N : door → dist
2:  N_c ← Leaf(s)
3:  PN_c ← the parent node of Leaf(s)
4:  while N_c ≠ N_s do
5:      CAD⊏(N_c) ← AD⊏(N_c)
6:      for d_i in AD⊏(N_c) do
7:          if TSD(s, d_i, t_c) / velocity + t_c ∉ d_i.ATIs then CAD⊏(N_c) \ d_i
8:          if d_i.d-type is PRD and d_i ∉ P2D(P(s)) then CAD⊏(N_c) \ d_i
9:      for each unmarked d ∈ AD⊏(PN_c) do
10:         mark d; compute TSD(s, d, t_c)
11:         if d ∈ AD⊏(N_s) then S2N[d] ← TSD(s, d, t_c)
12:     N_c ← PN_c; PN_c ← the parent node of PN_c
13: return S2N
14: function TSD(pt, d, t_c)
15:     return the cached result if computed
16:     N_d ← d's current corresponding node
17:     if pt is a door then return N_d.DC[pt, d, t_c]
18:     if N_d is a leaf node then
19:         if d ∈ P(pt) then return |pt, d|_E
20:         else
21:             D_o ← obtain doors that one can leave P(pt) from pt at t_c
22:             if D_o ≠ ∅ then
23:                 return min_{d_j ∈ D_o} (|pt, d_j|_E + TSD(d_j, d, t_c))
24:             else
25:                 return ∞
26:     else
27:         CN_d ← N_d's child node that contains pt
28:         return min_{d_j ∈ CAD⊏(CN_d)} (TSD(pt, d_j, t_c) + TSD(d_j, d, t_c))
```

Function TSD (lines 14–28) computes the shortest distance as follows. If the dis-

tance was previously computed, it just returns the cached result (line 15). If pt is a door, it obtains the corresponding node N_d of d (line 16), and directly obtains the shortest distance from pt to d from the distance cube (line 17). Otherwise, pt is a point. Two different cases are discussed. If N_d is a leaf node that means pt and d are in the same leaf node. In such a case, it returns the Euclidean distance between pt and d if they are in the same partition (line 19). If pt and d are not in the same partition, `TSD` first validates each possible door of the current partition if the door is still open when one gets it from pt, and then adds the valid ones in a set D_o. If D_o is not empty, the shortest distance is computed as the minimum of the sum of the distance from pt to a door $d_j \in D_o$ and $\texttt{TSD}(d_j, d, t_c)$. Otherwise, the shortest distance is returned as ∞. If N_d is a non-leaf node (line 26), we decompose the shortest distance into two parts: one from pt to an access door d_j in N_d's child node, and the other from d_j to d. Both parts are recursively computed by calling `TSD` (lines 27–28). In our implementation, we keep and share all intermediate results in the recursive calling of `TSD` to speed up the overall distance computation.

Algorithm 14 `Get_Dist_D2E`(d, e, t_c)

Require: A point s, a point e, and current time t_c
Ensure: The shortest distance from d to e at t_c

1: $N_c \leftarrow$ children of $N_t \cap$ ancestors of $\texttt{Leaf}(e)$
2: $CN_c \leftarrow$ children of $N_c \cap$ ancestors of $\texttt{Leaf}(e)$
3: **while** $CN_c \neq \texttt{Leaf}(e)$ **do**
4: **for** $d_i \in AD_{\sqsupset}(N_c)$ **do**
5: **if** $\texttt{TSD2}(d, d_i, t_c) / velocity + t_c \notin d_i.ATIs$ or $d_i.d\text{-}type$ is PRD **then**
6: $CAD_{\sqsubset}(N_c) \setminus d_i$
7: **for** each unmarked $d_j \in AD_{\sqsupset}(CN_c)$ **do**
8: mark d_j; $\texttt{TSD2}(d, d_j, t_c)$
9: $N_c \leftarrow CN_c$; $CN_c \leftarrow$ children of CN_c and ancestors of $\texttt{Leaf}(e)$
10: **return** $\texttt{TSD2}(d, e, t_c)$
11: **function** $\texttt{TSD2}(d, pt, t_c)$
12: **return** the cached result if computed
13: $N_d \leftarrow d$'s corresponding node
14: **if** pt is a door **then return** $N_d.\mathsf{DC}[d, pt, t_c]$
15: **if** N_d is a leaf node **then**
16: **if** $d \in \mathsf{P}(pt)$ **then return** $|d, pt|_E$
17: **else**
18: $D_o \leftarrow$ obtain doors that one can enter $\mathsf{P}(pt)$ from d at t_c
19: **if** $D_o \neq \varnothing$ **then**
20: **return** $\min_{d_j \in D_o}(\texttt{TSD2}(d, d_j, t_c) + |d_j, pt|_E)$
21: **else**
22: **return** ∞
23: **else**
24: $CN_d \leftarrow N_d$'s child node that contains pt
25: **return** $\min_{d_j \in CAD_{\sqsupset}(CN_d)} \left(\texttt{TSD2}(d, d_j, t_c) + \texttt{TSD2}(d_j, pt, t_c)\right)$

`Get_Dist_D2E` (Algorithm 14) returns the shortest distance (line 10) from an access door d of N_t to a point e for a current time t_c. The idea here is similar to that of Algorithm 13. The difference is that it starts from the current door d and searches in the direction towards the terminal point e. At the beginning, the current node N_c is set to N_t's child node that contains $\texttt{Leaf}(e)$, and CN_c is set to N_c's child node that contains $\texttt{Leaf}(e)$ (lines 1–2). In each step of processing N_c, each leavable access door d_i of N_c is checked if it is closed upon arrival or it is a private door (lines 4–6), and each enterable access door d_j of CN_c is processed to compute the shortest distance from d to d_j. By iteratively computing and caching the distances between the leavable access doors of N_c and the enterable access doors of CN_c, the algorithm can finally return the distance from d to e in a recursive manner (line 10).

Function `TSD2` (lines 11–25) computes the shortest distance from an access door d to a point pt (either an access door or a point e) with respect to current time t_c. Its processing is similar to `TSD` in Algorithm 13 but the direction is reversed such that the search starts from a door to a point in the leaf node.

Note that Algorithms 13 and 14 only compute the shortest distance, whereas the corresponding shortest path can also be constructed by keeping the last hop of each visited door. We omit the details due to the page limit.

Example C.4.2 (An Example of IT-INDEX based ITSPQ Processing)
Assuming the same ITSPQ from p_1 to p_2 as in Example C.3.1, we first find the host nodes of p_1 and p_2 in IT-INDEX, i.e., N_1 and N_6, respectively. Second, we find N_{LCA} of N_1 and N_6 as N_9, and then find the children of N_9, i.e., N_7 (the ancestor of N_1) and N_8 (the ancestor of N_6). Third, we find the paths from p_1 to each available access door of N_7, i.e., (p_1, d_3) with length 7m, (p_1, d_5, d_9, d_{13}) with length 19m, and (p_1, d_5, d_9, d_{14}) with length 22m. We do not consider (p_1, d_6, d_7) as it goes through a private partition v_6. Fourth, we find all paths from one available access door of N_8 to p_2, i.e, (d_3, p_2) with length 7m, (d_7, p_2) with length 2m, (d_{14}, p_2) with length 13m, and (d_{11}, d_{18}, p_2) with length 13m. Finally, we concatenate each path found in the third step and each path found in the fourth step. As a result, we return (p_1, d_3, p_2) with the shortest overall length 14m.

C.4.3 Complexity Analysis

Let $\texttt{V}$ be the total partition number, $\texttt{D}$ the total door number, $\texttt{V}_\texttt{o}$ the number of open partitions at a time point, $\texttt{D}_\texttt{o}$ the number of open doors at a time point, $\texttt{d}$ the average door number per partition, and $\texttt{w}$ the average number of doors on a shortest path. Let $\texttt{f}$ be the fan-out of IT-INDEX nodes, $\texttt{L}$ the number of leaf nodes, ρ the average number of access doors per node, and $\texttt{T}$ the average number of life intervals per door.

The space complexity of IT-GRAPH is $\mathcal{O}(\texttt{V}+\texttt{Vd}+\texttt{Vd}^2+\texttt{D}) = \mathcal{O}(\texttt{Vd}^2)$. The space complexity of IT-INDEX is $\mathcal{O}(\rho\texttt{DT})+ (\rho\texttt{f})^2\texttt{LT})$. Specifically, $\mathcal{O}(\rho\texttt{DT})$ captures the

space cost of the distance cubes in leaf nodes, whereas $\mathcal{O}((\rho \mathtt{f})^2 \mathtt{LT})$ is that of the distance cubes in non-leaf nodes. For a non-leaf node, $\rho \mathtt{f}$ corresponds to the number of access doors from a child node and $\mathtt{L}$ reflects the number of non-leaf nodes.

The time complexity of ITG/S is $\mathcal{O}(\mathtt{V} \log \mathtt{D} + \mathtt{w})$. It consists of the distance computing cost $\mathcal{O}(\mathtt{V} \log \mathtt{D})$ and the cost of backtracking the shortest path in $\mathtt{w}$ hops. The time complexity of ITG/A is generally $\mathcal{O}(\mathtt{V_o} \log \mathtt{D_o} + \mathtt{w})$. The difference between ITG/S and ITG/A is that ITG/A only considers the open doors/partitions in the current reduced graph instance. The time complexity of ITI is $\mathcal{O}(\rho^2 \log_{\mathtt{f}} \mathtt{L} + \mathtt{w} \log_{\mathtt{f}} \mathtt{L})$. Specifically, $\mathcal{O}(\rho^2 \log_{\mathtt{f}} \mathtt{L})$ refers to the cost of searching for the lowest common ancestor and finding a pair of access doors from that ancestor node, and $\mathcal{O}(\mathtt{w} \log_{\mathtt{f}} \mathtt{L})$ refers to the cost of constructing the shortest path.

C.5 Experimental Studies

Using both synthetic and real data, we evaluate the cost of constructing IT-INDEX (see Section C.4.1) and the search efficiency of our proposed methods ITG/S, ITG/A and ITI (see Fig. C.5). All experiments are implemented in Java and run on a PC with a 2.30GHz Intel i5 CPU and 16 GB memory.

C.5.1 Results on Synthetic Data

Settings

Indoor Space. Based on a real-world floorplan [12], we generate a multi-floor indoor space where each floor takes 1368m $\times$ 1368m. The irregular hallways are decomposed into smaller but regular partitions[2]. As a result, we obtain 141 partitions and 224 (virtual) doors that connect these partitions. We treat each room partition in the floorplan as a private partition and each hallway or staircase partition as a public partition. Consequently, on each floor, we have 53 public partitions and 88 private partitions. We duplicate the floorplan 3, 5, **7**, 9 or 11 times to simulate different indoor spaces. Every two adjacent floors are connected by four staircases, each having a stairway 20m long. In the default setting, we use a 7-floor indoor space with 987 partitions and 1568 doors.

Temporal Variations. We generate the ATIs for each door as follows. First, we crawl the online shop information of five shopping malls in Hong Kong, China, and parse the open and close times of those shops. We select random pairs of open time and close time to form the checkpoint set T in size of 4, **8**, 12, or 16. We then select a temporal door ratio (TDR) (20%, 40%, **60**%, 80% or 100%) of doors to be the *varied doors* that open and close from time to time. For each such temporally varying door, we assign it with up to three ATIs, each corresponding to a pair of open time and close time selected from T. The remaining doors are always open.

[2]The decomposition algorithm is given in [8].

Query Instances. Given a parameter s2t that controls the indoor distance from the start point p_s to the target point p_t, we generate query instances of ITSPQ(p_s, p_t, t) as follows. First, we randomly select a point p_s from the indoor space. Second, we find a door d whose indoor distance to p_s approximates s2t. Then, we expand from d to find a random point p_t whose indoor distance to p_s approaches s2t. For each setting of s2t, we generate five pairs of p_s and p_t to form the query instances. In each query instance, query time t is fixed to 12:00 to make a fair comparison. We also study the effect of using different values of t in query processing. The results are to be reported in Section C.5.1. Table C.3 lists the parameter settings in our experiments, where the default values are shown in bold.

Table C.3: Parameter Settings for Synthetic Data

Parameters	Settings
Floor Number	3, 5, **7**, 9, 11
$\|T\|$	4, **8**, 12, 16
TDR (% of varied doors)	20%, 40%, **60%**, 80%, 100%
s2t (m)	1100, 1300, **1500**, 1700, 1900
t	0:00, 2:00, ..., **12:00**, ..., 22:00

Baseline Method. We use a general temporal graph (GTG) [13–16] to form a baseline. Each vertex in GTG represents a door labeled with door type and active time intervals, and the weight of each edge is the distance between two doors. This way results in many door-to-door edges for the same partition and leads to large size of the graph. We adapt the synchronous check to GTG.[3] We may capture door directionality in a GTG's node as partition pairs, each implying that one can leave a partition to enter the other via the corresponding door. As this leads to considerably more space cost and search time cost, we assume all doors are bidirectional in the comparative experiments.

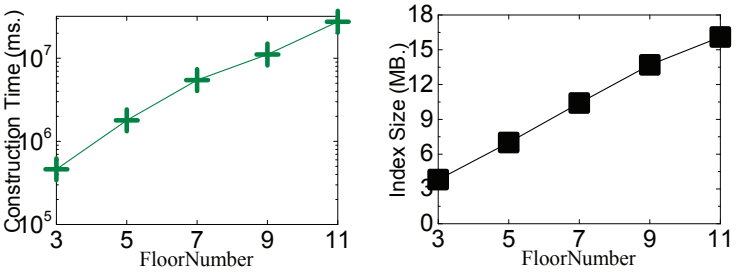
Performance Metrics. For IT-INDEX, we measure its construction time and index size. To compare the efficiency of different search algorithms, we run each query instance ten times, and measure the *average* running time, memory cost, and the number of door visits (NDV) per run of a single query instance.

Cost of Index Construction

In the default parameter setting, IT-GRAPH can be built within 310 ms and its size is around 3.5 MB, while IT-INDEX can be built within 30 minutes and its size is around 7 MB. The main cost of constructing IT-INDEX results from building the distance cubes associated with its tree nodes. Next, we vary and test different parameters in the performance evaluation of IT-INDEX construction. As the construction of IT-GRAPH is relatively steady in different parameter settings, we omit its evaluation result.

[3]Our preliminary experiments found that the difference of the searches using synchronous and asynchronous checks on GTG is similar to that on ITG. Therefore, we omit the GTG variant using asynchronous check.

Effect of Number of Floors. We vary the number of floors and report the corresponding construction time and index size in Figs. C.6(a) and C.6(b), respectively. When the number of floors increases, more doors and partitions will be involved in the indoor space. On the one hand, more partitions lead to more nodes in the tree structure of IT-INDEX. On the other hand, the number of doors contained by a tree node will also increase, which results in more time and space consumption for maintaining the distance cube of the tree node. Because of these two factors, both index construction time and index size increase steadily with an increasing number of floors. Nevertheless, when the number of floors increases to 11, the size of IT-INDEX is only 16.1 MB and the time of index construction is around 7.65 hours.

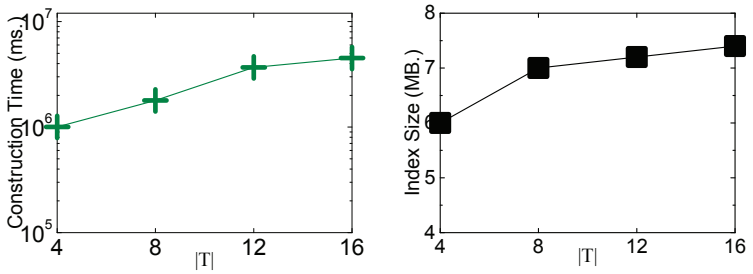


(a) Time vs. Floor (b) Size vs. Floor

Fig. C.6: Effect of Number of Floors

Effect of $|T|$. With other parameters fixed to default, we vary the checkpoint set size of T and report the time cost and size of the index construction in Figs. C.7(a) and C.7(b), respectively. Clearly, both IT-INDEX's time and space consumption increase moderately when $|T|$ increases. A larger $|T|$ results in more diversified door ATIs and more active temporal variations of indoor topology. In such a case, the construction of a distance cube may need to involve more life intervals on its time dimension, incurring larger memory cost and corresponding computation time. However, this trend flattens when $|T|$ grows at 12. At this point, the distance cube has maintained enough life interval information, and therefore increasing the number of checkpoints in T will not bring a significant lift in the index construction costs. For a large $|T| = 16$, IT-INDEX of 7.4 MB can be built within 1.25 hours.

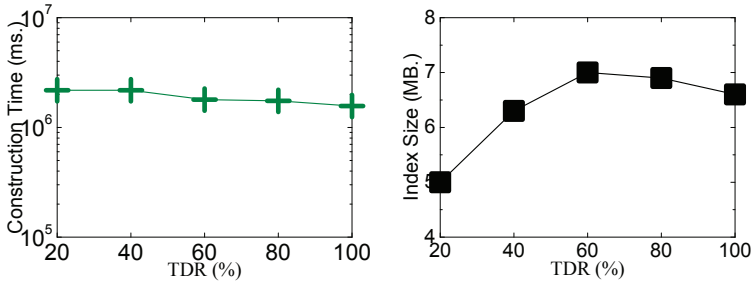
Effect of TDR. We also measure the time and memory costs of the index construction for different values of TDR. Referring to Fig. C.8(a), the index construction time is insensitive to an increasing TDR. Since the doors with temporal variations are randomly picked out from the space, the topology in a local range is not significantly affected by an increasing TDR. As the index is constructed based on the shortest path search within each local node, the construction time only changes slightly. On the other hand, the index size in Fig. C.8(b) increases first and then decreases when TDR becomes larger. In the beginning, the increase in TDR diversifies the temporal variations of doors. Consequently, the number of life intervals in the distance cube



(a) Time vs. T (b) Size vs. T

Fig. C.7: Effect of $|T|$

increases. When the TDR is increased at a large rate, doors may open and close more frequently at different times, resulting in that more pairs of doors in the distance cube correspond to empty records. Therefore, the index size decreases instead when TDR is larger than 60%.



(a) Time vs. TDR (b) Size vs. TDR

Fig. C.8: Effect of TDR

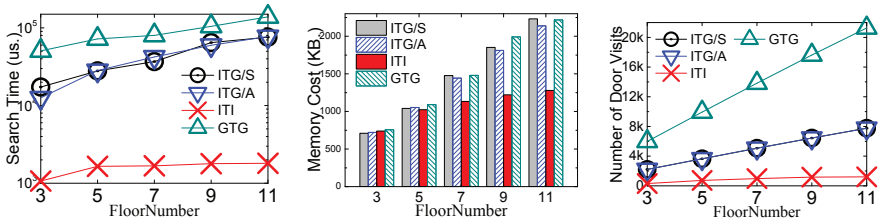
Summary. Overall, the index construction time and index size increase when the floor number and $|T|$ increase. However, the index construction cost is insensitive to the ratio of doors with temporal variations. The index size is sometimes even reduced when most doors are associated with temporal variations. These findings disclose that our IT-INDEX design is effective at indexing indoor venues with temporal variations.

Efficiency of Search Methods

We investigate the search time and memory cost of our proposed methods (ITG/S, ITG/A, and ITI) and the baseline method (GTG) under different parameter settings.

Effect of Number of Floors. Referring to Fig. C.9(a), GTG is the slowest due to its large graph size, whereas ITI always outperforms the others by an order of magnitude. Compared to its alternatives that iterate on the IT-GRAPH for the shortest

path search, ITI can quickly construct a set of local shortest paths stored in the relevant distance cubes of IT-INDEX. The precomputed results in IT-INDEX help reduce the overhead of the online search even when involving dynamic indoor topology changes. When the floor number increases, the search time of ITI only increases slightly, whereas that of ITG/S and ITG/A increases very rapidly. A larger floor number leads to more partitions and doors in a more complex IT-GRAPH structure, thus incurring more execution time for the two methods to explore the next hop door based on graph topology. In contrast, ITI only needs to search IT-GRAPH for a small number of path junctions (i.e., the access doors) when necessary. The explanation also applies to the trends of different methods' NDVs reported in Fig. C.9(c). Referring to the memory consumption in Fig. C.9(b), ITI uses the least memory because it does not require additional memory space for graph search. When the floor number is up to 5, the memory cost of ITG/A is slightly higher than that of ITG/S as ITG/A needs to maintain multiple versions of IT-GRAPH corresponding to different checkpoints. However, when the floor is greater than 5, ITG/A's memory cost is smaller than ITG/S's because ITG/S has to search a much more complex complete graph in this case. GTG requires more memory than ITG/S and ITG/A because it visits more doors (i.e., nodes in its graph).

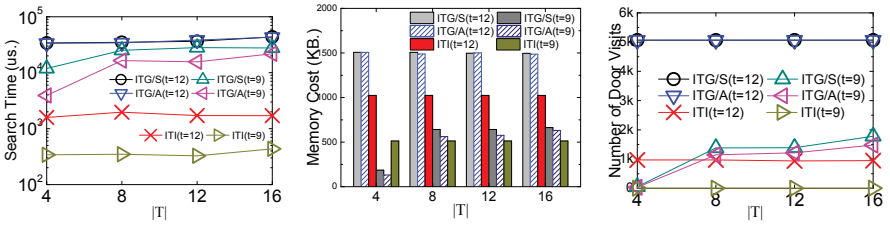


(a) Time vs. Floor (b) Memory vs. Floor (c) NDV vs. Floor

Fig. C.9: Effect of Number of Floors

Effect of $|T|$. Referring to Fig. C.10(a), the search time of each method is insensitive to $|T|$ when query time t is fixed to 12:00, a time nearly all doors in the space are open. In such a case, adding more checkpoints to T has little impact on the graph topology at query time. We add a group of tests with t fixed to 9:00. At this time, varying $|T|$ makes the size of active doors different, impacting the cost of graph search. Nevertheless, ITI still outperforms the other two by an order of magnitude. Referring to Fig. C.10(b), the memory cost of ITI is stable whenever at 12:00 or 9:00. When $|T|$ is 4, ITI cost more memory than ITG/S and ITG/A at 9:00. In this setting, there is nearly no route for the query instances at 9:00 with many doors closed, so ITG/S and ITG/A cost a few memory, whereas ITI stores the distance cube which leads to more memory cost than others. The NDV in Fig. C.10(c) exhibit trends consistent with those in the search time reported in Fig. C.10(a). Our experiments show that GTG always performs the worst when varying $|T|$. We exclude GTG in Figs. C.10(a), C.10(b)

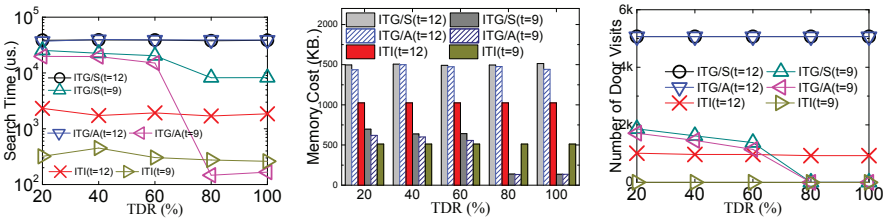
and C.10(c) to avoid distraction.



(a) Time vs. T (b) Memory vs. T (c) NDV vs. T

Fig. C.10: Effect of $|T|$

Effect of TDR. Referring to Figs. C.11(a) and C.11(c), the search time and NVD of each method are stable for t = 12:00 in different settings of TDR. Compared to the testing for t = 12:00, the search time and NVD for t = 9:00 decline because the topology are reduced. When TDR increases to 80%, there is nearly no routes for the query instances because more door are closed. In this case, ITG/A costs less time than ITG/S due to reduced topology. Still, ITI performs best in terms of the search time and NVD whenever at 12:00 and 9:00. Referring to Fig. C.11(b), ITI's memory cost is less than ITG/S and ITG/A. However, when TDR = 80% or 100% at 9:00, ITI costs more memory than others because it stores the distance cube, while ITG/S and ITG/A just cost a few memory because it expands a few doors.

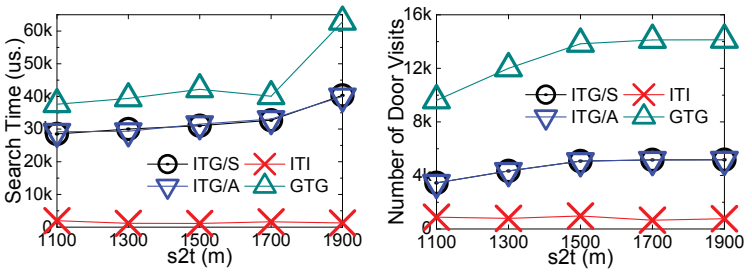


(a) Time vs. TDR (b) Memory vs. TDR (c) NDV vs. TDR

Fig. C.11: Effect of TDR

Effect of s2t. When we increase s2t, each method's search time increases slightly, as shown in Fig. C.12(a). A similar trend is seen for the NVDs reported in Fig. C.12(b). Nevertheless, ITI can still be several times faster than the other two, showing that the IT-INDEX is very efficient in the shortest path search using the pre-stored door-to-door information.

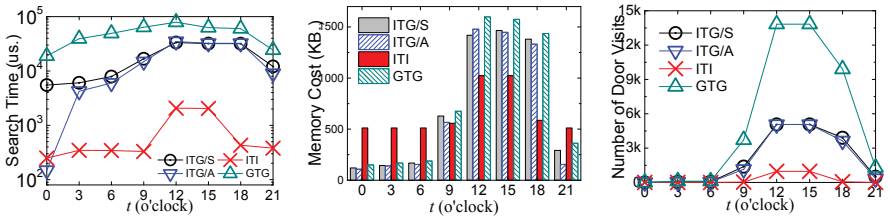
Effect of t. We also test the search methods' performance at different query times (t) in a day. Referring to Fig. C.13(a), the search time of each method increases before t comes to 9:00 and then decreases when t is over 18:00. In our setting, a large number



(a) Time vs. s2t (b) NDV vs. s2t

Fig. C.12: Effect of s2t

of doors have been closed for the time before 9:00 or after 18:00, and the corresponding graph, i.e., IT-GRAPH or GTG, becomes simpler due to the reduced temporal variations. On the contrary, the graph structure becomes more complex when more doors are open during the period from 12:00 to 15:00. A complex graph structure costs ITG/A, ITG/S and GTG more time to search for accessible doors. The time cost of ITI increases as well but more slowly, as it only searches for a small set of access doors that connect the local shortest paths. Referring to Fig. C.13(b), ITI's memory cost before 9:00 is the highest as it needs to maintain an additional IT-INDEX. Between 12:00 and 15:00, the memory costs of all methods stay constant because nearly all doors are open and the indoor topology is relatively stable. After 18:00, the memory costs of all methods decrease as the graph structure becomes simpler. Referring to Fig. C.13(c), after 9:00, NDV grows rapidly for ITG/S and ITG/A, especially for GTG because more doors are open. In contrast, ITI's NDV only increases very slightly as it only needs to explore several access doors during its search.



(a) Time vs. t (b) Memory vs. t (c) NDV vs. t

Fig. C.13: Effect of t

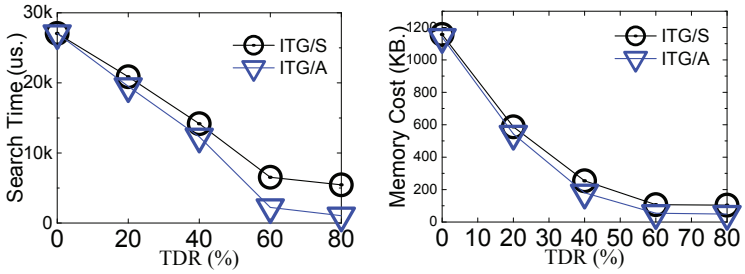
Summary. In general, ITI always has the highest search efficiency with the aid of IT-INDEX. It is faster than the other three by an order of magnitude in most tests. The search time and memory cost of ITI increase slowly when the graph topology becomes more complex, whereas those of ITG/S, ITG/A and GTG increase rapidly

as these methods rely heavily on the graph search. Moreover, GTG performs the worst due to its large graph size.

Comparison of ITG/A and ITG/S

We scrutinize the difference between ITG/A and ITG/S. Two parameters in our setting determines the number of close and open doors, namely the temporal door ratio TDR and the checkpoint set size $|T|$. We use the control variates method, which stipulates that T is an empty set such that all temporal doors controlled by TDR keep closed. This reduces the fluctuation of query search time due to uneven distribution of ATIs. As a result, we can analyze the impact of varying TDR on ITG/A and ITG/S.

We conduct the comparative experiments under the default settings and report the results in Figs. C.14(a) and C.14(b). As we can see, both measures of ITG/A and ITG/S decrease steadily with an increasing TDR. However, when TDR increases to 80%, i.e., 80% doors are closed at a query time, ITG/A shows great advantages over ITG/S in both search time and memory consumption. In general, if there are many doors with temporal variations in the space, ITG/A is more efficient because it involves search in only several reduced graph instances maintained asynchronously, without expensive on-the-fly door checks over the full topology graph. Therefore, for usual scenarios without urgencies like a fire, we recommend ITG/A with asynchronous checks.

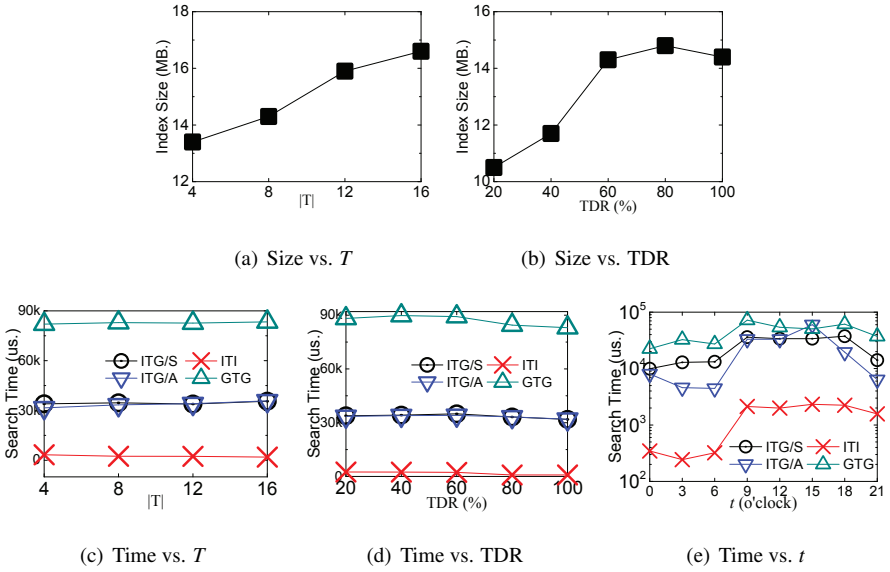


(a) Time vs. TDR (b) Memory vs. TDR

Fig. C.14: Comparison of ITG/A and ITG/S

C.5.2 Results on Real Data

We collect a dataset with real indoor topology and temporal variation information from a seven-floor, 2700m $\times$ 2000m shopping mall in Hangzhou, China. There are ten staircases in which each stairway between two adjacent floors is roughly 20m long. In our setting, we treat all stores and equipment rooms as private partitions, and hallways and staircases as public partitions. As a result, we obtain 497 private and 553 public partitions connected by 2093 doors. We set the default values of $|T|$ and



(a) Size vs. T (b) Size vs. TDR

(c) Time vs. T (d) Time vs. TDR (e) Time vs. t

Fig. C.15: Result on Real Data

TDR as 8 and 60%, respectively, according to the real-world use of the mall. It takes around 3.53 hours to construct IT-INDEX in a size of 14.3 MB for the whole space. We randomly select five pairs of p_s and p_t such that the distances between each pair is roughly s2t $= 1500m$. The default query time t is fixed to 12:00.

Effect of $|T|$ on Index Size. We modify the checkpoint set T by adding to or removing pairs of open and close times from it. Fig. C.15(a) implies that the index size increases moderately with a larger T. When $|T|$ is 16, the size of IT-INDEX is 16.6 MB. The indoor space in the real data contains more doors and partitions. Thus, the index size is larger than that in the synthetic data.

Effect of TDR on Index Size. We also modify the fraction of doors with temporal variations in the real data from 20% to 100%. Referring to Fig. C.15(b), the results are consistent with the counterparts reported in Fig. C.8(b). The index sizes increase first as TDR increases. When TDR grows to a certain extent, the temporal variations of doors tend to be consistent and less life interval information needs to be maintained in the distance cubes.

Effect of $|T|$ on Search Time. We also investigate different methods' search efficiency by varying $|T|$. The search time is reported Fig. C.15(c). ITI outperforms others significantly, whereas GTG performs the worst in terms of the search time. The search time of each method is insensitive to $|T|$ as the queries are issued at 12:00, a time nearly all doors are open.

Effect of TDR on Search Time. Referring to Fig. C.15(d), all methods' search time stays stable as a complete graph is used at default query time 12:00. ITI can

return the shortest path in less than 3ms, clearly outperforming its alternatives. Still, GTG runs several times slower than the others.

Effect of t on Search Time. Referring to Fig. C.15(e), the search time of each method increases before t comes to 9:00, stays stable until t comes to 18:00, and then goes down after that. Note that the search time of ITI only increases slightly when more doors are added to the graph structure (from 9:00 to 18:00). This indicates that ITI is not significantly affected by the change of graph topology. Similar trend can be seen in ITG/S, ITG/A and GTG, but GTG always needs more time than the others.

C.6 Related Work

Indoor Spatial Queries. Indoor spaces feature multiple entities like doors, walls, and rooms, altogether forming a complex topology that complicates distance-aware queries. Becker et al. [17] propose an indoor symbolic model with semantic descriptions for indoor entities and study the route planning problem based on the proposed model. Li et al. [18] propose a lattice-based semantic location model that keeps the semantic relationships and distance in each location-exit lattice to support the navigation in indoor spaces. Yuan et al. [19] propose a model to construct a wayfinding network that is based on the geometry of the indoor space and that supports length-dependent optimal routing. Goetz and Zipf [20] define a weighted indoor routing graph with semantic information to create a detailed and user-adaptive model for route search. Lu et al. [7] propose a distance-aware indoor space model and an indexing framework to facilitate distance-aware queries. To speed up distance-aware indoor path finding, Shao et al. [9] design IP-tree and VIP-tree that enable more aggressive pruning. VIP-tree also supports indoor trip planning based on neighbour expansion [3]. Luo et al. [21] study the time-constrained sequenced route query (TCSRQ) in indoor space. The result of TCSRQ considers the stay-time period and types of indoor locations. Alamri et al. [22] propose a cell-based index structure (C-tree) to group and manage updates of indoor moving objects based on hop counts. As it does not support indoor walking distances, C-tree cannot apply to the shortest path problem studied in this paper. Many other indoor spatial queries [4, 8, 23, 24] such as range queries and kNN queries have been also studied for indoor spaces. However, none of these works consider temporal variation information associated with indoor entities, and thus they all fall short in solving the problem studied in this paper.

Temporal Graph Queries. Temporal variations have been considered on graph structure in which the connections between vertices are active at specific times [25]. Huo et al. [13] analyze and evaluate shortest-path queries on evolving social graphs. Semertzidis et al. [14] study the historical reachability queries on evolving graphs. In the same setting, Semertzidis et al. [15] study three general types of historical queries, namely, historical graph queries, historical time queries and historical top-k queries. Semertzidis et al. [26] use a compressed time neighbourhood and path index to find the durable matches of an input pattern on the temporal graphs. Akiba et al. [27] study

the shortest-path distance queries on large time-evolving graphs by using two dynamic indexing schemes. Huang et al. [16] investigate the properties of temporal DFS and BFS, and propose efficient algorithms for route query in a temporal graph. Hirsch et al. [28] propose a method for routing of information over dynamic communication networks. These works, mainly oriented to social graphs or communication networks, can support the shortest path query by setting the cost as the edge weight.

There are also some temporal graph queries specific to physical spaces. Ding et al. [29] propose time-dependent algorithms to find the minimum-travel-time path from a start point p_s to an end point p_e with the best departure time relative to the current query time t_c. Ardakani et al. [30] propose an adaptive approach to solve the dynamic shortest path problem. In the same setting, Ardakani et al. [31] design an A$*$ algorithm using the decremental approach to speed up the shortest path query processing in dynamic networks. Wei et al. [32] propose an efficient distance and path oracle on dynamic road networks using the randomization technique.

However, these aforementioned techniques cannot resolve ITSPQ directly due to two reasons. First, those social-graph oriented works [13–16, 26–28] do not support the impact of travel time in the dynamic graph search, i.e., when an object arrives at a node (a door in our setting) at a particular time, the node may already be invalid. As a matter of fact, the existing techniques make use of a static snapshot of the evolving graph for query processing. Second, none of the aforementioned techniques [29–32] consider the indoor semantic information (e.g., private/public partitions and doors) that further complicates the query processing. We might model the indoor building as the aforementioned general temporal graph (time-evolving graph), i.e., all doors are modeled as nodes labeled with type (private or public) and active time intervals, and each edge is labeled with distance. However, this way falls short in our problem setting. It fails to represent the door directionality information directly. Also, such a general temporal graph results in many door-to-door edges for the same partition, which will render the graph based search inefficient.

C.7 Conclusion

In this paper, we study the shortest path queries for indoor venues with temporal variations. Given a start point p_s, a target point p_t, and a current time t, an indoor temporal-variation aware shortest path query ITSPQ(p_s, p_t, t) returns the valid shortest path from p_s to p_t. We present a set of techniques to answer ITSPQ efficiently. First, we propose a graph structure (IT-GRAPH) that integrates the indoor temporal variations of doors into the indoor topology and design a Dijkstra-based framework to answer ITSPQ with IT-GRAPH. Under the framework, two different versions of algorithms are devised to check doors accessibility synchronously and asynchronously. Furthermore, we propose an index structure (IT-INDEX) that extends the state-of-the-art index significantly by storing dynamic door-to-door distances in a compact distance cube associated with tree nodes. An IT-INDEX based algorithm is also devised

to answer ITSPQ. The extensive experiments demonstrate that our IT-INDEX based method is the most efficient for processing ITSPQ as it materializes partial but critical shortest distance information in the corresponding tree nodes.

For future work, it is relevant to consider allowing waiting times for the doors to open while minimizing the total time to the destination. It is interesting to support other practical issues such as service time and capacity of elevators. Also, it is useful to support other query types (e.g., range query and distance-aware join) using our indoor temporal-variation aware structures.

Acknowledgement

This work was supported by Independent Research Fund Denmark (No. 8022-00366B), Australian Research Council (No. FT180100140 and DP180103411), Hong Kong RGC Projects (No. 12200817 & C6030-18GF), and Guangdong Basic and Applied Basic Research Foundation (No. 2019B1515130001).

References

[1] M. A. Cheema, "Indoor location-based services: challenges and opportunities," *SIGSPATIAL Special*, vol. 10, no. 2, pp. 10–17, 2018.

[2] H. Li, H. Lu, L. Shou, G. Chen, and K. Chen, "Finding most popular indoor semantic locations using uncertain mobility data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 11, pp. 2108–2123, 2018.

[3] Z. Shao, M. A. Cheema, and D. Taniar, "Trip planning queries in indoor venues," *The Computer Journal*, vol. 61, no. 3, pp. 409–426, 2018.

[4] X. Xie, H. Lu, and T. B. Pedersen, "Distance-aware join for indoor moving objects," *IEEE Transactions on Knowledge and Data Engineering*, vol. 2, no. 27, pp. 428–442, 2015.

[5] H. Li, H. Lu, L. Shou, G. Chen, and K. Chen, "In search of indoor dense regions: An approach using indoor positioning data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 8, pp. 1481–1495, 2018.

[6] Z. Feng, T. Liu, H. Li, H. Lu, L. Shou, and J. Xu, "Indoor top-k keyword-aware routing query," in *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 2020, pp. 1213–1224.

[7] H. Lu, X. Cao, and C. S. Jensen, "A foundation for efficient indoor distance-aware query processing," in *2012 IEEE 28th International Conference on Data Engineering*. IEEE, 2012, pp. 438–449.

[8] X. Xie, H. Lu, and T. B. Pedersen, "Efficient distance-aware query evaluation on indoor moving objects," in *2013 IEEE 29th International Conference on Data Engineering (ICDE)*. IEEE, 2013, pp. 434–445.

[9] Z. Shao, M. A. Cheema, D. Taniar, and H. Lu, "Vip-tree: an effective index for indoor spatial queries," *Proceedings of the VLDB Endowment*, vol. 10, no. 4, pp. 325–336, 2016.

[10] T. Liu, Z. Feng, H. Li, H. Lu, M. A. Cheema, H. Cheng, and J. Xu, "Shortest path queries for indoor venues with temporal variations," in *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 2020, pp. 2014–2017.

[11] https://en.wikipedia.org/wiki/Walking.

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

[13] W. Huo and V. J. Tsotras, "Efficient temporal shortest path queries on evolving social graphs," in *Proceedings of the 26th International Conference on Scientific and Statistical Database Management*, 2014, pp. 1–4.

[14] K. Semertzidis, E. Pitoura, and K. Lillis, "Timereach: Historical reachability queries on evolving graphs." in *EDBT*, vol. 15, 2015, pp. 121–132.

[15] K. Semertzidis and E. Pitoura, "Time traveling in graphs using a graph database." in *EDBT/ICDT Workshops*, 2016, p. 96.

[16] S. Huang, J. Cheng, and H. Wu, "Temporal graph traversals: Definitions, algorithms, and applications," *arXiv preprint arXiv:1401.1919*, 2014.

[17] C. Becker and F. Dürr, "On location models for ubiquitous computing," *Personal and Ubiquitous Computing*, vol. 9, no. 1, pp. 20–31, 2005.

[18] D. Li and D. L. Lee, "A lattice-based semantic location model for indoor navigation," in *The Ninth International Conference on Mobile Data Management (mdm 2008)*. IEEE, 2008, pp. 17–24.

[19] W. Yuan and M. Schneider, "inav: An indoor navigation model supporting length-dependent optimal routing," in *Geospatial thinking*. Springer, 2010, pp. 299–313.

[20] M. Goetz and A. Zipf, "Formal definition of a user-adaptive and length-optimal routing graph for complex indoor environments," *Geo-Spatial Information Science*, vol. 14, no. 2, pp. 119–128, 2011.

[21] W. Luo, P. Jin, and L. Yue, "Time-constrained sequenced route query in indoor spaces," in *Asia-Pacific Web Conference*. Springer, 2016, pp. 129–140.

[22] S. Alamri, D. Taniar, K. Nguyen, and A. Alamri, "C-tree: efficient cell-based indexing of indoor mobile objects," *Journal of Ambient Intelligence and Humanized Computing*, pp. 1–17, 2019.

[23] B. Yang, H. Lu, and C. S. Jensen, "Probabilistic threshold k nearest neighbor queries over moving objects in symbolic indoor space," in *Proceedings of the 13th international conference on extending database technology*, 2010, pp. 335–346.

[24] W. Yuan and M. Schneider, "Supporting continuous range queries in indoor space," in *2010 Eleventh International Conference on Mobile Data Management*. IEEE, 2010, pp. 209–214.

[25] O. Michail, "An introduction to temporal graphs: An algorithmic perspective," *Internet Mathematics*, vol. 12, no. 4, pp. 239–280, 2016.

[26] K. Semertzidis and E. Pitoura, "Top-*k* durable graph pattern queries on temporal graphs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 1, pp. 181–194, 2018.

[27] T. Akiba, Y. Iwata, and Y. Yoshida, "Dynamic and historical shortest-path distance queries on large evolving networks by pruned landmark labeling," in *Proceedings of the 23rd international conference on World wide web*, 2014, pp. 237–248.

[28] M. J. Hirsch, A. Sadeghnejad, and H. Ortiz-Peña, "Shortest paths for routing information over temporally dynamic communication networks," in *MILCOM 2017-2017 IEEE Military Communications Conference (MILCOM)*. IEEE, 2017, pp. 587–591.

[29] B. Ding, J. X. Yu, and L. Qin, "Finding time-dependent shortest paths over large graphs," in *Proceedings of the 11th international conference on Extending database technology: Advances in database technology*, 2008, pp. 205–216.

[30] M. K. Ardakani and L. Sun, "Decremental algorithm for adaptive routing incorporating traveler information," *Computers & operations research*, vol. 39, no. 12, pp. 3012–3020, 2012.

[31] M. K. Ardakani and M. Tavana, "A decremental approach with the A* algorithm for speeding-up the optimization process in dynamic shortest path problems," *Measurement*, vol. 60, pp. 299–307, 2015.

[32] V. J. Wei, R. C.-W. Wong, and C. Long, "Architecture-intact oracle for fastest path and time queries on dynamic spatial networks," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, 2020, pp. 1841–1856.

Paper D

Towards Crowd-aware Indoor Path Planning

Tiantian Liu, Huan Li, Hua Lu, Muhammad Aamir Cheema, Lidan Shou

The paper has been published in
Proceedings of the VLDB Endowment, 14(8), pp. 1365–1377, 2021.

Abstract

Indoor venues accommodate many people who collectively form crowds. Such crowds in turn influence people's routing choices, e.g., people may prefer to avoid crowded rooms when walking from A to B. This paper studies two types of crowd-aware indoor path planning queries. The Indoor Crowd-Aware Fastest Path Query (FPQ) finds a path with the shortest travel time in the presence of crowds, whereas the Indoor Least Crowded Path Query (LCPQ) finds a path encountering the least objects en route. To process the queries, we design a unified framework with three major components. First, an indoor crowd model organizes indoor topology and captures object flows between rooms. Second, a time-evolving population estimator derives room populations for a future timestamp to support crowd-aware routing cost computations in query processing. Third, two exact and two approximate query processing algorithms process each type of query. All algorithms are based on graph traversal over the indoor crowd model and use the same search framework with different strategies of updating the populations during the search process. All proposals are evaluated experimentally on synthetic and real data. The experimental results demonstrate the efficiency and scalability of our framework and query processing algorithms.

D.1 Introduction

Indoor route planning queries are among the most fundamental queries underlying indoor location-based services (LBS) [1–5]. Such queries can facilitate people in need. For example, in an airport or a train station, passengers prefer to find the fastest path from their current position to the boarding gate. In addition to the shortest or fastest paths, indoor routing supports many variations that meet practical needs. For instance, customers in a shopping mall would like to find a path that can cover some given keywords like a coffee shop and shoes [6]. Meanwhile, indoor venues accommodate many people who collectively form crowds that may in turn influence people's routing choices. For example, crowds may influence one's moving speed, which will have an effect on the travel time of a path. In some places like an airport where passengers are sensitive to travel time, a topologically shortest path may still incur the too long time and result in missing flight if the path fails to consider the effect of crowds. In other scenarios, people en route may prefer to encounter fewer people. For example, during the COVID-19 pandemic, people would like to find a path to avoid human contact as much as possible. As another example, autonomous objects (e.g., driverless cars in an airport) also prefer a path with fewer people en route to mitigate the interference and inconvenience caused by contact with people.

In this paper, we formulate and study two crowd-aware indoor path planning queries. Referring to Fig. D.1, given a source point p_s, a target point p_t, and a query time t, an Indoor Crowd-Aware Fastest Path Query (FPQ) returns a path with the shortest travel time in the presence of crowds, whereas an Indoor Least Crowded Path

Query (LCPQ) returns a path that encounters the least objects en route. As an indoor path is essentially a series of indoor partitions (basic topological units like rooms), FPQ's routing cost is partition-passing time, whereas an LCPQ's is partition-passing contact.

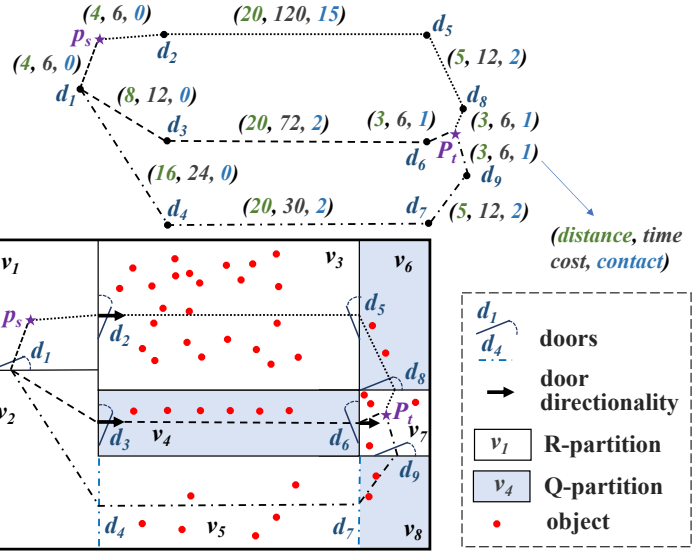


Fig. D.1: An Example of Floorplan at Query Time t_q

We consider two types of indoor partitions. A *Queue Partition* (Q-partition) requests objects to enter and exit in a line, e.g., a security-check line in an airport or a ticketing entrance in a theater. A *Random Partition* (R-partition) refers to a more general case where there is no restriction on how to pass the partition but one's movement slows down when encountering a crowd. Due to the different topological natures of the Q-partitions and R-partitions, partition-passing time for FPQ and partition-passing contact for LCPQ should be defined differently for the two partition types.

Existing techniques cannot handle the novel FPQ and LCPQ. First, techniques for outdoor route planning [7–14] do not work for indoor spaces with distinct entities like doors, walls, and rooms, altogether forming a complex topology. Second, the existing indoor route planning methods [1, 2, 6, 15] do not consider the effect of crowds, lacking the modeling foundation for FPQ and LCPQ. Third, some works study indoor flow and density [16, 17], but they do not touch upon route planning.

To solve FPQ and LCPQ efficiently, we design a crowd-aware query processing framework. The framework is composed of three layers. First, the indoor crowd model is the foundation layer of the framework. The model can handle three kinds of information, namely indoor topology, indoor geometry, and crowd-evolution. A time-evolving population estimator derives the future flows and populations for indoor partitions. The estimated values are used as basic routing costs in FPQ and LCPQ.

The query processing layer consists of two parts. In part one, two functions, namely partition-passing time function and partition-passing contact function, calculate the routing cost for FPQ and LCPQ, respectively. Again, the differences in routing costs for different partition types are unified into a single computing process. On top of that, part two provides two exact and two approximate path search algorithms that each can process both query types. One of the exact searches uses a global estimator, whereas the other uses an estimator that only estimates a partition's population by looking up its upstream partitions' flows. The two approximate algorithms speed up query processing at the cost of query accuracy. One of them only derives populations for partial partitions, and the other derives populations only when necessary. All proposed techniques are experimentally evaluated on synthetic and real data. The experimental results demonstrate the efficiency and scalability of the proposed framework and query processing algorithms. The results also show the two approximate search algorithms achieve good routing accuracy.

This paper makes the following key contributions.

- We formulate Indoor Crowd-Aware Fastest Path Query (FPQ) and Indoor Least Crowded Path Query (LCPQ), and propose a unified processing framework for these queries (Section D.2).
- We design an indoor crowd model that organizes indoor topology and captures indoor partition flows and densities (Section D.3).
- We devise a time-evolving population estimator to derive future time-dependent flows and populations for partitions (Section D.4).
- We design two exact and two approximate query processing algorithms that each can process both query types (Section D.5).
- We conduct extensive experiments on synthetic and real datasets to evaluate our proposals (Section D.6).

In addition, Section D.7 reviews the related work and Section D.8 concludes the paper.

D.2 Preliminaries

Table D.1 lists the notations frequently used in this paper.

D.2.1 Indoor Crowds

An indoor space is divided by walls and doors into *indoor partitions*. A **Queue Partition** (Q-partition) requests objects to enter and leave *sequentially*, while a **Random Partition** (R-partition) has no such a restriction and objects can enter and leave it *randomly*. Note that the type of partition is usually fixed and will change only when the

Table D.1: Notations

Symbol	Meaning
v, d, p	Partition, door, and indoor point
o, O	Object, object set
C	Indoor crowd
$t_{o \triangleright C}$, $t_{C \triangleright o}$	The times o joins and leaves C
$RT(d_i)$	The sequence of d_i's report timestamps
$UTI(v_k)$	The set of v_k's unit (update) time intervals
$\delta_{t_x,t_{x+1}}(v_k)$	v_k's density over $[t_x,t_{x+1}]$
$\rho(v_k,t_c)$	v_k's lagging coefficient at time t_c
$T(d_i,d_j,v_k,t_c)$	The time to pass v_k from d_i to d_j at t_c
$\kappa(d_i,d_j,v_k,t_c)$	The object contact to pass v_k from d_i to d_j at t_c

space layout is redesigned. The issue of topological change is out of the scope of this paper.

Within a partition, moving objects (e.g., persons) may form a crowd. Corresponding to the partition types, we formally define an *indoor crowd* as follows.

Definition 1 (Indoor Crowd). *An indoor crowd $C_{t_s,t_e}(v_k)$ [1] is a set of moving objects in a partition v_k during a certain time interval $[t_s,t_e]$. $C.\tau$ denotes the type of crowd C.*

1) *In a **Queue Crowd** (Q-crowd), objects join and leave the crowd in the first-in-first-out (FIFO) manner. Formally, $\forall o_i,o_j \in C_k \wedge C_k.\tau = \mathsf{Q}, t_{o_i \triangleright C_k} \leq t_{o_j \triangleright C_k} \Rightarrow t_{C_k \triangleright o_i} \leq t_{C_k \triangleright o_j}$, where $t_{o_i \triangleright C_k}$ and $t_{C_k \triangleright o_i}$ is the time o_i joins and leaves C_k, respectively.*

2) *In a **Random Crowd** (R-crowd), objects join and leave the crowd without any ordering restrictions. Formally, $\exists o_i,o_j \in C_k \wedge C_k.\tau = \mathsf{R}, t_{o_i \triangleright C_k} < t_{o_j \triangleright C_k}, t_{C_k \triangleright o_i} \geq t_{C_k \triangleright o_j}$.*

A crowd changes as objects join and leave from time to time. In other words, the object population and density of a partition are time-varying, rendering an object's routing cost passing the partition to change as well. Therefore, for crowd-aware routing, it is of fundamental importance to know a crowd's dynamic population or density. This demands dynamic data from a localization system.

However, a localization system may not record the exact trajectory or join/leave time of each individual object due to computing/storage limitations and location privacy concerns. Alternatively, a system may maintain the current number of objects in each partition (or a crowd) and records the number of objects joining and leaving during a time interval. This can be easily achieved, e.g., by installing a counter at a door. In our setting, each door counter reports objects' joining and leaving at a predefined frequency. This means that the object numbers in a crowd are updated at a number of discrete timestamps. Specifically, we use a time-ordered sequence $RT(d_i) = (t_{i1},\ldots,t_{in})$ to denote the **report timestamps** of the counter at door d_i. As a result, the **update timestamps** relevant to a partition v_k is a time-ordered sequence

[1] When time is not of particular interest, we use C_k to denote v_k's associated crowd.

$UT(v_k) = \bigcup_{d_j \in P2D(v_k)} RT(d_j)$ where $P2D(v_k)$ refers to all doors of partition v_k. Each two consecutive timestamps in the sequence $UT(v_k)$ forms an **unit (update) time interval**. The set of all such intervals from $UT(v_k)$ is denoted by $UTI(v_k)$.

At the routing query time, it is necessary to know the flows in the future. However, future exact object numbers from door counters are unavailable at that moment. To this end, we employ door flow functions to model the crowd-evolution (detailed in Section D.3.2).

We define a partition's *time-parameterized density* as follows.

Definition 2 (Time-Parameterized Density). *Given a partition v_k and its unit time interval $[t_x, t_{x+1}] \in UTI(v_k)$, its* time-parameterized density *over $[t_x, t_{x+1}]$ is $\delta_{t_x,t_{x+1}}(v_k) = |C_k|/Area(v_k)$, where $|C_k|$ is v_k's population over $[t_x, t_{x+1}]$ and $Area(v_k)$ is v_k's area.*

The population and density in this paper are time-parameterized unless mentioned otherwise. A partition v_k's density at an arbitrary timestamp t_c is estimated with respect to the unit time interval covering t_c. Specifically, we have $\delta_{t_c}(v_k) = \delta_{t_x,t_{x+1}}(v_k)$ where $t_x \leq t_c < t_{x+1}, [t_x, t_{x+1}] \in UTI(v_k)$.

D.2.2 Problem Formulation

In an indoor routing problem, a basic step is to move from one door to another through their in-between partition. To measure the cost to pass a partition, the intra-partition **door-to-door distance** [6] for two doors d_i and d_j is

$$d2d(d_i, d_j) = \begin{cases} |d_i, d_j|_E, & \text{if } D2P_{\sqsupset}(d_i) \cap D2P_{\sqsubset}(d_j) \neq \varnothing; \\ \infty, & \text{otherwise.} \end{cases} \tag{D.1}$$

where $D2P_{\sqsupset}(d_i)$ 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. Therefore, $D2P_{\sqsupset}(d_i) \cap D2P_{\sqsubset}(d_j) \neq \varnothing$ means d_i and d_j share a common partition that one can enter via d_i and leave via d_j. In this case, the Euclidean distance is used between d_i and d_j. Otherwise, the distance between them is set to infinite.

Definition 3 (Indoor Path). *An indoor path from the source p_s to the target p_t is $\phi = (p_s, d_x, \ldots, d_y, p_t)$, where $(d_x, \ldots, d_y)$ is a door sequence, d_x is a leaveable door of p_s's host partition, d_y is an enterable door of p_t's host partiton, and each two consecutive doors d_n, d_{n+1} $(x \leq n < y)$ on ϕ have $D2P_{\sqsupset}(d_n) \cap D2P_{\sqsubset}(d_{n+1}) \neq \varnothing$. Each two consecutive path nodes form a* path segment. *The distance of ϕ is computed as $dist_\phi = |p_s, d_x|_E + \sum_{n=x}^{y-1} d2d(d_n, d_{n+1}) + |d_y, p_t|_E$.*

When there is no crowd, the basic time cost of passing an in-between partition v_k from d_i to d_j can be estimated based on the average object moving speed $\bar{s}$, i.e.,

$T^{(b)}(d_i,d_j) = d2d(d_i,d_j)/\bar{s}$. To reflect a crowd's impact, we use the **lagging coefficient** $\rho(v_k,t_c)$ that takes into account the crowd's density and type as follows.

$$\rho(v_k,t_c) = \begin{cases} 1+e^{\delta_{t_c}(v_k)/\mathsf{D}_k^{max}}, & \text{if } C_k.\tau = \mathsf{Q}; \\ 1+e^{(\delta_{t_c}(v_k)/\mathsf{D}_k^{max})^2}, & \text{otherwise.} \end{cases} \tag{D.2}$$

where $\delta_{t_c}(v_k)$ is v_k's density at time t_c and D_k^{max} corresponds to the maximum density[2] of v_k. For a Q-partition, the ratio $\delta_{t_c}(v_k)/\mathsf{D}_k^{max}$ is applied to reflect the crowding degree. We modify the speed-density model [18] to calculate the lagging coefficient in Equation D.2 which reflects real-world scenarios, e.g., in common sense, a crowd usually impacts people's moving speed and results in longer travel time. Equation D.2 guarantees that the coefficient is always greater than 1 and it increases monotonically as v_k's density increases. For an R-partition, the square of the ratio is used because R-crowds incur less lagging effect.

Note that other forms of lagging coefficients can be defined and supported within our framework, e.g., lagging can be multiplied by the object number for a queue crowd. Since the lagging coefficient is *not* our research focus, we simply apply Equation D.2 in this study.

Using the lagging coefficient, we can calculate our crowd-aware and time- dependent **partition-passing time** as follows.

$$T(d_i,d_j,v_k,t_c) = T^{(b)}(d_i,d_j) \cdot \rho(v_k,t_c) \tag{D.3}$$

An object needs longer time to pass a more crowded partition.

As a special case, we replace d_i with p_s or replace d_j with p_t in Equation D.3, to estimate the cost of a path segment starting with p_s or ending with p_t. Accordingly, v_k is the host partition of p_s or p_t.

With the partition-passing time, we can plan the fastest indoor path for users to avoid undesirable congestion caused by indoor crowds. An indoor path ϕ's *overall travel time* T_ϕ is computed as the sum of the time of passing the partition between each path segment on ϕ. The fastest path query problem is defined as follows.

Problem 1 (Indoor Crowd-Aware Fastest Path Query `FPQ`**).** *Given a source p_s and a target p_t, an* indoor crowd-aware fastest path query $FPQ(p_s,p_t,t)$ *returns a path $\phi(p_s,d_i,\ldots,d_j,p_t)$ such that a) the overall travel time T_ϕ is minimized and b) ϕ is the shortest among all satisfying a). Formally, $\nexists\phi' \neq \phi$, $T_{\phi'} \leq T_\phi \wedge dist_{\phi'} < dist_\phi$.*

Note that the partition-passing time is determined by the time one arrives at that partition, while the arrival time, in turn, is dependent on the partition-passing time of the previous partition. This calls for on-the-fly computation during the search to obtain the overall travel time T_ϕ, which is to be detailed in Section D.5.

[2]The maximum capacity (and therefore the maximum density) of a partition is usually known, such as the room capacity for fire safety.

Example D.2.1 (An Example of Indoor Crowd-Aware Fastest Path Query) Fig. D.1 illustrates an indoor space at time t_q. The query time and crowd-evolution snapshot are considered. We indicate the distance, partition-passing time and object contact on each path segment in the top sketch. We suppose that there are some events in v_7, and v_4, v_6 and v_8 are Q-partitions for ID check before entering v_7. Given a query $\texttt{FPQ}(p_s, p_t, t_q)$, there are three candidate paths, namely $\phi_1(p_s, d_2, d_5, d_8, p_t)$, $\phi_2(p_s, d_1, d_3, d_6, p_t)$, and $\phi_3(p_s, d_1, d_4, d_7, d_9, p_t)$. Only considering the distance but not the impact from crowds, ϕ_1 is the shortest with a length of 32 meters, while those of ϕ_2 and ϕ_3 are 35 meters and 48 meters, respectively. However, ϕ_1 is not expected to be the fastest path when crowds are concerned. To be specific, ϕ_1 goes through a highly crowded R-partition v_3, incurring a total travel time of 144 seconds. For ϕ_2, the low-populated Q-partition v_4 with a long queue is involved, making the total time cost be 96 seconds. Among all, ϕ_3 is expected to be the fastest with an overall cost of 78 seconds, though it is the longest distance passing 5 partitions.

Another practically interesting problem is to find the shortest path that contacts the least objects. E.g., it is useful to find a path that avoids human contact as much as possible in the COVID-19 case. Given a path segment (d_i, d_j) that goes through a partition v_k, we calculate the **partition-passing contact** as follows.

$$\kappa(d_i, d_j, v_k, t_c) = \begin{cases} (|d_i, d_j|_E \cdot \texttt{w}) \cdot \delta_{t_c}(v_k), & \text{if } C_k.\tau = \mathsf{R}; \\ (\texttt{w} / |d_i, d_j|_E) \cdot (\delta_{t_c}(v_k) \cdot Area(v_k)), & \text{otherwise.} \end{cases} \tag{D.4}$$

Given a partition v_k, its enterable door d_i, and its leaveable door d_j, for any object reaching d_i at time t_c, the partition-passing contact to pass v_k and reach d_j is defined in terms of the number of objects covered by the buffer of the path segment. The contact to pass an R-partition is the partition density multiplied by the buffer area that is approximated as $|d_i, d_j|_E \cdot \texttt{w}$ where $\texttt{w}$ is the buffer width. The contact to pass a Q-partition is the objects within the $\texttt{w}$ long queue line centered at the user's position, i.e., the proportion $\texttt{w}/|d_i, d_j|_E$ of the total objects in the queue. This reflects common sense. For example, if we pass a random crowd, the close contacts are those who we meet in the buffer width. If we pass a queue crowd, we only have close distance with those in front of or behind us.

In our implementation, we set $\texttt{w}$ as the unit distance of 1m. For example, many countries suggest people keep a physical distance of 1m in the COVID-19 pandemic. Similar to the computation of the overall travel time T_ϕ, an indoor path ϕ's *overall contact* κ_ϕ is computed as the sum of the partition-passing contacts of path segments on ϕ. Likewise, Equation D.4 applies to the path segment starting with p_s and ending with p_t. Accordingly, we formulate the least crowded path query as follows.

Problem 2 (Indoor Least Crowded Path Query LCPQ**).** *Given a source p_s and a target p_t, an* indoor least crowded path query *LCPQ(p_s, p_t, t) returns a path $\phi(p_s, d_i, \ldots, d_j, p_t)$ such that a) the overall contact is the least, and b) ϕ is the shortest among all satisfying a). Formally, $\nexists \phi' \neq \phi$, $\kappa_{\phi'} \leq \kappa_\phi \wedge dist_{\phi'} < dist_\phi$.*

Example D.2.2 (An Example of Indoor Least Crowded Path Query) Consider a query LCPQ(p_s, p_t, t_q) in Fig. D.1, the candidates ϕ_1, ϕ_2 and ϕ_3 involve 18, 3 and 5 contacts from the partitions which they pass, respectively. The query returns ϕ_2 since it contacts the fewest objects.

D.2.3 Solution Framework

We propose a crowd-aware query processing framework as illustrated in Fig. D.2.

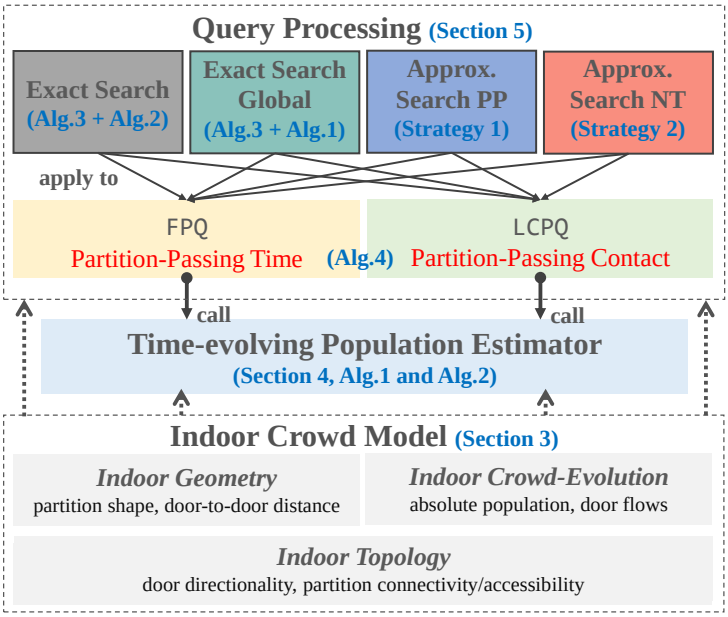


Fig. D.2: Crowd-Aware Path Planning Framework

In the bottom, an indoor crowd model (cf. Section D.3) maintains the following aspects of an indoor space: *Indoor Topology* that captures the directionality of doors and connectivity/accessibility of partitions, *Indoor Geometry* that records the shapes of partitions and walking distances between two doors, and *Indoor Crowd-Evolution* that models the objects joining and leaving the crowds.

Enabled by the indoor crowd model, a *time-evolving population estimator* in the middle layer derives populations (and densities) of partitions at a future time and pro-

vides them to the query algorithms. The population estimation process will be detailed in Section D.4.

In the top layer, crowd-aware search algorithms process FPQ and LCPQ. Both algorithms are based on graph traversal over the indoor crowd model. To expand to the next path node with the minimum cost, FPQ's algorithm estimates the partition-passing time, while LCPQ search algorithm estimates the partition-passing contact. Both costs are estimated based on the time-evolving populations derived in the middle layer. For both queries, two exact and two approximate search algorithms are proposed. Their main difference lies in the strategy of updating population(s) during the search. All search algorithms will be presented in Section D.5. Thanks to modular construction, our framework can be easily extended or reduced. For example, to support regular path planning, we only need the components Indoor Topology and an appropriate query processing algorithm that can be a variant of Algorithm 17.

D.3 Indoor Crowd Model

D.3.1 Model Structure

As an extension of the accessibility graph [19], the *indoor crowd model* is a directed, labeled graph $G(V, E, L_V, L_E)$ where

1) V is the set of vertices, each for an indoor partition.

2) E is the set of directed edges such that each edge $e(v_i, v_j, d_k) \in E$ means one can reach v_j from v_i through a door d_k, i.e., $v_i \in D2P_{\sqsubset}(d_k)$ and $v_j \in D2P_{\sqsupset}(d_k)$.

3) L_V is the set of vertex labels. Each label in L_V is attached to a partition and captured as a five tuple $[v_i, Area(v_i), \mathsf{M}_{d2d}, \tau, (\mathsf{P}^i_{t_l}, t_l)]$. In particular, v_i identifies the associated partition, $Area(v_i)$ is v_i's area, M_{d2d} is a matrix that stores the intra-partition distance (See Equation D.1) between each pair of doors of v_i. In addition, $\tau \in \{\mathsf{R}, \mathsf{Q}\}$ indicates the type of v_i's crowd and $(\mathsf{P}^i_{t_l}, t_l)$ means that v_i's absolute population at a latest timestamp t_l is known as $\mathsf{P}^i_{t_l}$. In practice, the model can record the populations at historical timestamps, though only the latest population is relevant to a query.

4) L_E is the edge label set. For an edge $(v_i, v_j, d_k) \in E$, its label consists of a **door flow function** $f(v_i, v_j, d_k)$ that models the dynamic object flows from v_i to v_j via d_k and a local array $\mathsf{F}[t]$ storing the actual object flows at each update timestamp t.

Figure D.3 depicts the indoor crowd model corresponding to the space in Figure D.1. Unlike a general time-dependent graph (GTG) [10, 20], our model represents doors as edges and partitions as vertices. A GTG may model doors as vertices and partitions as edges and capture time-varying populations or distances as edge weights, but this way falls short in solving our problem. First, GTG's vertices fail to capture

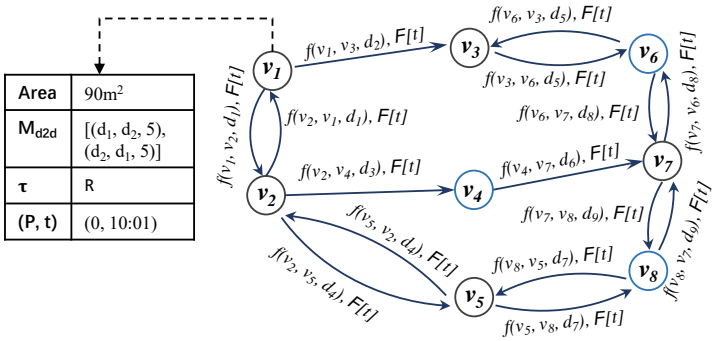


Fig. D.3: An Example of Indoor Crowd Model

the door directionality (e.g., unidirectional security check doors) directly. Referring to Figure D.1, d_2 is unidirectional such that one can only go through d_2 from v_1 to v_3. In a GTG, the edges cannot be directed because each edge connects two doors and one can always go from one door to any other door in the same room. E.g., one cannot go through d_2 from v_3 to v_1, but she can go to any door in v_1 from d_2 if she is in v_1. The directionality information can be added in each node, e.g., that for node d_2 can be $\{(v_1, v_3)\}$, and that for node d_1 can be $\{(v_1, v_2), (v_2, v_1)\}$. However, it will result in considerably more space and search costs. Second, a GTG will result in many door-to-door edges for the same partition, which will render the graph-based search inefficient. The experimental comparison with GTG is reported in Section D.6.

The time-evolving function $f(v_i, v_j, d_k)$ models the number of objects flowing from v_i to v_j at each report time interval of d_k. In practice, it can be implemented as a time-series prediction model driven by historical data such as ARIMA [21] and LSTM [22], or it can be approximated by a queueing distribution function. For the ease of presentation, in this paper, we employ a specific queueing distribution function to predict the door flows (Section D.3.2). Nevertheless, the door flow function can be replaced by other appropriate models or functions, which entails no change to any of the other parts in the overall computation framework (Figure D.2).

D.3.2 Door Flow Function

Following the classic Poisson distribution in queueing theory [23], we design the following door flow function:

$$f(v_i, v_j, d_k) : t \mapsto \mathsf{P}_t, t \in RT(d_k), \mathsf{P}_t \sim Poisson(\lambda) \tag{D.5}$$

where $t \in RT(d_k)$ is a report timestamp of d_k, P_t is the population that flows from v_i to v_j between t and d_k's next report timestamp, and λ is the expected value of P_t under Poisson distribution.

The door flow function is parameterized by λ and fitted based on a recent period

of historical records in a format of $(t', \mathsf{P}_{t'})$. In practice, for each door counter, the most recent timestamps' flows can be accessed from the local array F in the graph edge. An independent thread estimates λ upon such most recent records. Note that the focus of this paper is not to estimate λ based on historical data. For its technical details, we refer readers to a previous work [24]. In our setting, at any query time, an up-to-date door flow function is ready to predict flows for future report timestamps.

D.4 Time-evolving Populations

D.4.1 Rectifying Door Flows

At a query time t_q, we can access a partition v_k's latest population $\mathsf{P}^k_{t_l}$ at an earlier time $t_l \leq t_q$ from the indoor crowd model. To enable the cost estimation for routing, we need to derive v_k's time-evolving population and its future inflows/outflows based on $\mathsf{P}^k_{t_l}$.

Let $[t_0, t_1] \in UTI(v_k)$ be the unit time interval covering t_l. We have $\mathsf{P}^k_{t_0,t_1} = \mathsf{P}^k_{t_l}$, meaning that v_k's population over $[t_0, t_1]$ is equal to $\mathsf{P}^k_{t_l}$. Subsequently, for a future unit time interval $[t_x, t_{x+1}] \in UTI(v_k)$, we compute its population as

$$\mathsf{P}^k_{t_x,t_{x+1}} = \mathsf{P}^k_{t_{x-1},t_x} - out(v_k,t_x) + in(v_k,t_x),\ x = 1,2,\ldots \tag{D.6}$$

where $out(v_k,t_x)$ and $in(v_k,t_x)$ are v_k's estimated outflow and inflow at update timestamp t_x, respectively.

Suppose that all relevant door flow functions are ready at t_q. The inflow and outflow at a future update timestamp can be directly estimated based on the expected values λ. Formally,

$$out(v_k,t_x) = \sum_{d_i \in P2D_{\sqsubset}(v_k) \wedge t_x \in RT(d_i)} \sum_{v_p \in D2P_{\sqsupset}(d_i)} f(v_k,v_p,d_i).\lambda$$

$$in(v_k,t_x) = \sum_{d_j \in P2D_{\sqsupset}(v_k) \wedge t_x \in RT(d_j)} \sum_{v_q \in D2P_{\sqsubset}(d_j)} f(v_q,v_k,d_j).\lambda$$

where d_i (resp. d_j) is a leaveable (resp. enterable) door updated at time t_x and $v_p \in D2P_{\sqsupset}(d_i)$ (resp. $v_q \in D2P_{\sqsubset}(d_i)$) is its enterable (resp. leaveable) partition.

However, the estimated flows may be contrary to the real situation such that a partition's current population ($\mathsf{P}^k_{t_{x-1},t_x}$ in Equation D.6) cannot satisfy its outflow ($out(v_k,t_x)$ in Equation D.6). In this case, flows at doors should be rectified.

A basic idea is to rectify the expected outflow at each step such that it is not larger than the partition v_k's current population. Meanwhile, v_k's inflow is naturally rectified as it is derived from the outflows of its adjacent partitions at the previous step. In general, a dependency exists between partitions. It demands a suitable way to rectify the relevant outflows at the update timestamps.

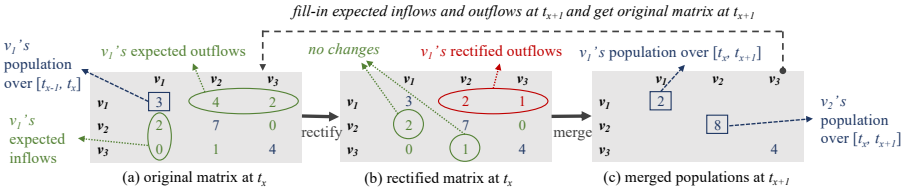


Fig. D.4: An Example of Rectifying Flows Globally

An example is depicted in Figure D.4, which rectifies the door flows globally. To ease the presentation, at each particular update timestamp t_x we put the absolute populations and door flows in a $|V| \times |V|$ matrix M, where $|V|$ corresponds to the total number of partitions. In particular, $\mathsf{M}[i,i]$ refers to partition v_i's absolute population over unit time interval $[t_{x-1}, t_x]$, while $\mathsf{M}[i,j]$ $(i \neq j)$ means the flow value from partition v_i to v_j over the next unit time interval $[t_x, t_{x+1}]$. Referring to Figure D.4(a), partition v_1's population over $[t_{x-1}, t_x]$ is 3 and that of v_2 is 7. Besides, v_1's expected outflows to v_2 and v_3 are 4 and 2, respectively; v_2's inflow from v_1 and v_3 are 4 and 1, respectively. Considering the space efficiency, in the implementation, we store the absolute populations on the graph nodes and the estimated flows on graph edges. That is, the space complexity at each update timestamp is $|V+E|$.

A rectification is then applied to each row of the original matrix as exemplified in Figure D.4(b). Specifically, v_1's current population (i.e., 3) is less than the summation of its subsequent outflows (i.e., $4+2=6$). In this case, we scale down the outflows at all doors to ensure that the actual number of objects outflowing is exactly equal to the current population. That is, $\mathsf{M}'[1,2] = \mathsf{M}[1,2] \cdot (3/6) = 2$ and $\mathsf{M}'[1,3] = \mathsf{M}[1,3] \cdot (3/6) = 1$, where M and M' represent the original and rectified matrix, respectively. Note that non-integer values may appear in the rectification. For the computation precision, we use non-integer values in the whole iterative derivation process. Intuitively, the values in the matrix are a probability estimate, i.e., how likely an object will appear in or move to a certain partition.

After the rectification, each partition's population over the next interval $[t_x, t_{x+1}]$ is computed based on Equation D.6. In particular, partition v_i's new population is obtained by deducting the overall outflows at the i-th row of M' and then adding the overall inflows at the i-th column of M'. Referring to Figure D.4(c), v_1's new population is $3-3+2=2$ while v_2's is $7-2+3=8$. After the merges on each partition, we fill in the matrix inflows and outflows at the timestamp t_{x+1}, and derive the populations iteratively.

D.4.2 Implementation of Population Estimator

This section presents two versions of population estimators. The *global estimator* estimates all partitions' populations globally (corresponding to the example in Fig-

ure D.4), whereas the *local estimator* only estimates a relevant partition's population by looking up its upstream partitions flows.

Algorithm 15 POPULATIONGLOBAL (future timestamp t^a, indoor crowd model G)

1: get the latest update timestamp t_l^G from G
2: $UT_G \leftarrow \bigcup_{d_j \in G.D} RT(d_j)$
3: $\mathsf{A} \leftarrow toArray(\{t_c \mid t_c \in UT_G \wedge t_c \geq t_l^G \wedge t_c \leq t^a\})$
4: **for** $t_c \in \mathsf{A}$ **do**
5: **for** $e(v_i, v_j, d_k) \in G.E$ **do**
6: **if** $t_c \in RT(d_k)$ **then** $e.\mathsf{F}[t_c] \leftarrow f(v_i, v_j, d_k).\lambda$
7: **else** $e.\mathsf{F}[t_c] \leftarrow 0$
8: **for** $v_i \in V$ **do**
9: **for** $d_k \in P2D_{\sqsubset}(v_i)$ **do**
10: $out_i \leftarrow out_i + (v_i, v_j, d_k).\mathsf{F}[t_c]$
11: get v_i's latest population record (P_t^i, t) from G
12: **if** $\mathsf{P}_t^i - out_i < 0$ **then**
13: **for** $d_k \in P2D_{\sqsubset}(v_i)$ **do**
14: $(v_i, v_j, d_k).\mathsf{F}[t_c] \leftarrow (v_i, v_j, d_k).\mathsf{F}[t_c] \cdot \mathsf{P}_t^i / out_i$
15: **for** $v_i \in V$ **do**
16: **for** $d_k \in P2D_{\sqsubset}(v_i)$ **do**
17: $out_i \leftarrow out_i + (v_i, v_j, d_k).\mathsf{F}[t_c]$
18: **for** $d_k \in P2D_{\sqcap}(v_i)$ **do**
19: $in_i \leftarrow in_i + (v_j, v_i, d_k).\mathsf{F}[t_c]$
20: $\mathsf{P}_{t_c}^i \leftarrow \mathsf{P}_t^i - out_i + in_i$; add $(\mathsf{P}_{t_c}^i, t_c)$ to $G.v_i$

The global estimator (Algorithm 15) takes the indoor crowd model G as input and derives the populations from the latest update timestamp in G to a future timestamp $t^a \geq t_q$. In the beginning, the globally latest update timestamp t_l^G over all partitions is obtained (line 1). Then, the set UT_G of all doors' report timestamps is obtained (line 2) and the period of interest is extracted out of UT_G and organized into an array A (line 3). The algorithm then progressively derives the populations for each timestamp in A (lines 4–20). For each timestamp $t_c \in \mathsf{A}$, the algorithm iterates on each edge $e(v_i, v_j, d_k)$. If the corresponding door d_k updates at t_c, e's flow value at t_c, i.e., $e.\mathsf{F}[t_c]$, is assigned with the estimated flow of the corresponding flow function (line 6). Otherwise, the flow value $e.\mathsf{F}[t_c]$ is assigned with 0 (line 7). Here $e.\mathsf{F}$ is a local array to maintain the rectified flow at each timestamp. Subsequently, the algorithm goes through each partition v_i and aggregates its expected outflow out_i (lines 8–10). If out_i is greater than the population P_t^i at the latest update timestamp, the estimated flows at edges need to be rectified following the example in Figure D.4 (lines 11–14). Afterwards, the current timestamp t_c's population for each partition is computed according to Equation D.6 and added to G (lines 15–20). Once the process is finished, all partitions' populations at each timestamp can be accessed from the edge nodes of G.

Algorithm 15 needs to update the populations for all partitions. This incurs much unnecessary computation since a path planning search at a particular time only concerns a number of relevant partitions and their populations. If these partitions' populations and flows can be precisely derived without a global update, the overall updating cost can be reduced substantially.

Algorithm 16 POPULATIONLOCAL (partition v_i, future timestamp t^a, indoor crowd model G)

1: get v_i's latest update timestamp t_l from G
2: $UT_G \leftarrow \bigcup_{d_j \in G.D} RT(d_j)$
3: $\mathsf{A} \leftarrow toArray(\{t_c \mid t_c \in UT_G \wedge t_c \geq t_l \wedge t_c \leq t^a\})$, $t_{max} \leftarrow \mathsf{A}.max()$
4: **while** A is not empty **do**
5: $\quad t_c \leftarrow \mathsf{A}.max()$; $\mathsf{A} \leftarrow \mathsf{A} \setminus t_c$
6: $\quad$ **if** $t_c = t_l$ **then**
7: $\quad\quad$ get v_i's latest population record $(\mathsf{P}^i_{t_l}, t_l)$ from G
8: $\quad\quad \mathsf{P}^i_{t_c} \leftarrow \mathsf{P}^i_{t_l}$
9: $\quad$ **else** $\mathsf{P}^i_{t_c} \leftarrow$ POPULATIONLOCAL (v_i, t_c, G)
10: $\quad$ **for** $d_k \in P2D_{\sqsubset}(v_i)$ **do**
11: $\quad\quad$ **if** $(v_i, v_j, d_k).\mathsf{F}[t_c]$ is null **then**
12: $\quad\quad\quad (v_i, v_j, d_k).\mathsf{F}[t_c] \leftarrow f(v_i, v_j, d_k).\lambda$
13: $\quad\quad out_i \leftarrow out_i + (v_i, v_j, d_k).\mathsf{F}[t_c]$
14: $\quad\quad$ **if** $\mathsf{P}^i_{t_c} - out_i < 0$ **then**
15: $\quad\quad\quad$ **for** $d_k \in P2D_{\sqsubset}(v_i)$ **do**
16: $\quad\quad\quad\quad (v_i, v_j, d_k).\mathsf{F}[t_c] \leftarrow (v_i, v_j, d_k).\mathsf{F}[t_c] \cdot \mathsf{P}^i_{t_c} / out_i$
17: $\quad\quad\quad out_i \leftarrow \mathsf{P}^i_{t_c}$
18: $\quad$ **for** $d_k \in P2D_{\sqsupset}(v_i)$ **do**
19: $\quad\quad$ **if** $(v_j, v_i, d_k).\mathsf{F}[t_c]$ is null **then**
20: $\quad\quad\quad$ POPULATIONLOCAL (v_j, t_c, G)
21: $\quad\quad in_i \leftarrow in_i + (v_j, v_i, d_k).\mathsf{F}[t_c]$
22: $\quad \mathsf{P}^i_{t_c} \leftarrow \mathsf{P}^i_{t_l} - out_i + in_i$
23: $t \leftarrow t_{max}$
24: **return** P^i_t

The local estimator is formalized in Algorithm 16. It derives a specific partition's population in a recursive manner. Its preparation (lines 1–3) is almost the same as the counterpart in Algorithm 15, except that the latest update timestamp t_l in line 1 is with respect to the input partition v_i.

Next, the algorithm derives v_i's population in reverse temporal order (lines 4–24). Specifically, the newest update timestamp t_c in A is archived and removed from A (line 5). If t_c just equals t_l, the population is directly obtained from G (lines 6–8). Otherwise, t_c is an earlier timestamp to t_l, and the algorithm recursively calls

Algorithm 16 to obtain v_i's population $\mathsf{P}^i_{t_c}$ at t_c (line 9).

Once $\mathsf{P}^i_{t_c}$ is derived, the expected flow from each upstream door flow function (see lines 10-12) is compared to v_i's population. Note that the intermediate results are maintained in each edge's local array $\mathsf{F}[t]$ to avoid repeated computations (line 11). If an expected outflow is larger than $\mathsf{P}^i_{t_c}$, a rectification is performed (lines 14–16). In this case, the rectified outflow is assigned with $\mathsf{P}^i_{t_c}$ (line 17). Then, the inflows from all enterable doors are also derived (lines 18-22). For each enterable door d_k, if its inflow has not been derived, Algorithm 16 is recursively called to get the adjacent partition v_j's population at time t_c (line 18-20). Note that the inflow from v_j to v_i will be rectified in this recursion. After that, v_i's overall inflow is obtained (line 21) and its population is computed (line 22). The last two lines of Algorithm 16 return the population nearest to the input time t^a.

Once the partition's population at a particular arrival time is derived, the corresponding partition-passing cost (time or contact) can be computed easily according to Equation D.3 or D.4. Both global and local population estimators can be utilized by the exact search presented in Section D.5.1 (see line 17 in Algorithm 17).

D.5 Query Processing Algorithms

D.5.1 Exact Algorithms for `FPQ` and `LCPQ`

On top of the indoor crowd model (Section D.3), we propose an indoor path search process in Algorithm 17. Following the spirit of Dijkstra's algorithm, our algorithm can handle both `FPQ` and `LCPQ`, as long as a cost measure corresponding to a specific query type is set as the routing cost of the graph traversal. Algorithm 17 returns an indoor path ϕ from the source p_s to the target p_t that satisfies the query type `QT` for a particular query time t_q.

The algorithm starts with initializing a priority queue Q (line 1) whose priority is the minimum travel time for `FPQ` and the minimum contact for `LCPQ`. Also, an array *prev* is initialized to record each path node's previous node in the search (lines 2–3). Then, the full set UT_G of the report timestamps over all doors are obtained (line 4). With respect to the query time t_q, the latest update timestamp in UT_G is found and assigned to t_l (line 5). Variable t_l is the latest population derivation time in the search. Next, the cost of the current search is initialized (line 6). The cost for `LCPQ` consists of overall travel distance, overall travel time, and overall contact value. The cost for `FPQ` only contains the first two. The source and initial cost are put into a stamp S_0, and S_0 is pushed into Q and maintained in an array A_S as well (lines 7–8).

After the preparation, the algorithm explores the next path node towards p_t in an order controlled by Q (lines 9–22). Specifically, the stamp S_i with the lowest cost is popped from Q, and the corresponding path node d_i is obtained (line 10). If d_i is p_t, i.e., the searched is complete, the algorithm calls a function GETPATH(p_t, *prev*, p_s) to return the reverse path from p_t to p_s (line 11). Otherwise, the algorithm explores the

Algorithm 17 SEARCH (source p_s, target p_t, query time t_q, indoor crowd model G, query type QT)

```
1: initialize a priority queue Q
2: for each door d_i ∈ G.D do prev[d_i] ← null
3: prev[p_s] ← null; prev[p_t] ← null
4: UT_G ← ⋃_{d_j ∈ G.D} RT(d_j)
5: t_l ← max{t ∈ UT_G | t ≤ t_q}; t^a ← ∅        ▷ latest update time t_l and arrival time t^a
6: if QT = LCPQ then cost_0 ← (0,0,0) else cost_0 ← (0,0)        ▷ (distance, time, contact) for LCPQ and (distance, time) for FPQ
7: S_0 ← (p_s, cost_0)                                          ▷ S(node, cost)
8: A_S[p_s] ← S_0; Q.push(S_0)
9: while Q is not empty do
10:     S_i ← Q.pop(); d_i ← S_i.node
11:     if d_i = p_t then return GETPATH(p_t, prev, p_s)
12:     if d_i = p_s then v ← host(p_s)
13:     else v ← D2P_□(d_i) \ d_i's previous partition
14:     mark d_i as visited
15:     t^a ← max{t ∈ UT_G | t ≤ t_q + S_i.cost.time}
16:     if t^a > t_l then                                       ▷ further derive populations
17:         POPULATION(t^a, G)
18:         t_l ← t^a
19:     if d_i ∈ P2D_□(host(p_t)) then
20:         EXPAND(d_i, p_t, G, v, t^a, S_i, QT)                ▷ towards target p_t
21:     for each unvisited door d_j ∈ P2D_⊏(v) do
22:         EXPAND(d_i, d_j, G, v, t^a, S_i, QT)
```

next path node as follows.

First, if the current node is the source p_s, the current partition v is obtained as the host of p_s (line 12). Otherwise, v is assigned as d_i's enterable partition[3] (line 13). Then, d_i is marked as visited (line 14). Next, the estimated cost to pass v is obtained as S_i.cost.time and it is added to the query time t_q to get the arrival time t^a to the next path node (line 15). An alignment to the update timestamps in UT_G is needed for t^a. The algorithm then determines if the population needs to be derived to meet the next arrival time t^a (lines 16–18). If the latest derivation time t_l is earlier than t^a, a population estimator is invoked to get v's derived populations up to t^a (line 17). Here, either the global (Algorithm 15) or local estimator (Algorithm 16) can be used. The performance difference of these two ways will be experimentally studied in Section D.6.

Afterward, it expands to the next node from the current node d_i. If d_i is an enterable door of p_t's host partition, the expansion goes towards p_t (lines 19-20). Regardless of

[3]To ease presentation, here we only show the case that a door connects two partitions. A complex space can be handled by maintaining a collection of enterable partitions.

whether the current path reaches p_t's host partition, the expansion should also reach each unvisited leaveable door of the current partition v (lines 21–22). This ensures that the planned path can leave and re-enter p_t's host partition when the host is currently too crowded.

The function EXPAND is formalized in Algorithm 18, which expands from the current node p_1 to the next possible node p_2 through partition v. First, it estimates the cost to reach p_2 from p_1 through an inline function COST (see lines 7–16). In particular, the distance between p_1 and p_2 is obtained as the door-to-door distance if both are doors, or Euclidean distance if either is an indoor point (lines 8–10). Then, the population of the partition v_k to pass is obtained from G (line 11), and the partition-passing time and contact are computed based on Equations D.3 and D.4, respectively (lines 12 and 14). The corresponding cost is then returned according to the query type `QT` (lines 13 and 15–16).

Back to EXPAND in Algorithm 18, once the cost to pass v is obtained, it is added to the current stamp S_i's cost to get the overall cost in the current expansion, i.e., $cost_c$ (line 1). The tuple-form cost is summed in an element-wise way. Next, the estimated cost to reach p_2 so far is obtained from the array A_S (line 2). The algorithm compares the current estimated cost $cost_c$ to the previously recorded cost $cost_{pre}$. If $cost_{pre}$ does not exist or $cost_c$ is lower, a valid expand is performed (lines 3–6). Specifically, a new stamp S' is formed with the next path node p_2 and the new cost $cost_c$. It is then pushed to Q. If an old stamp exists with the same node, the old stamp is updated by S' (line 5). Then, S' is inserted into A_S and p_2's previous path node is recorded as p_1.

In the exact search, the time-evolving populations are rectified and computed rigidly timestamp by timestamp. This may result in a bottleneck in the graph traversal. We intend to reduce the workload for population derivation by approximation. On the one hand, the severity of population derivation can be skipped for those less important partitions, e.g., those far away from the current partition to pass. On the other hand, some of the update timestamps in the iterative derivation can be skipped if the population changes within that iteration period is relatively stable. We proceed to introduce two approximate search algorithms for `FPQ` and `LCPQ`.

D.5.2 Approximate Algorithms for `FPQ` and `LCPQ`

We design two strategies to derive approximate populations.

Strategy 1: Population Derivation for Partial Partitions (`PP`). Recall that the population derivation in Algorithm 16 (see line 20) recursively obtains its adjacent partition's population to ensure the overall derivation is fully precise. This recursion terminates when the outflows of all relevant partitions at all relevant update timestamps have been rectified. In fact, the door flows from a long distance or at a very old timestamp only have a slight impact on a partition's current population. Therefore, one option is to rectify only the outflows of the current relevant partition without strictly processing the outflows of its upstream partitions (i.e., the inflows to the current relevant partition). To this end, only a minor change is made to Algorithm 16: line 20

Algorithm 18 EXPAND (start node p_1, end node p_2, indoor crowd model G, partition v, arrival time t^a, stamp S_i, query type QT)

1: $cost_c \leftarrow S_i.\text{cost} + \text{COST}(p_1, p_2, v, t^a, G)$ ▷ element-wise
2: $cost_{pre} \leftarrow \mathsf{A}_S[p_2].\text{cost}$
3: **if** $cost_{pre}$ is *null* or $cost_c < cost_{pre}$ **then**
4: $S' \leftarrow (p_2, cost_c)$
5: $\mathsf{Q}.push(S')$ ▷ update if exists
6: $\mathsf{A}_S[p_2] \leftarrow S'$; $prev[p_2] \leftarrow p_1$
7: **function** COST(p_1, p_2, v_k, t^a, G)
8: $dist \leftarrow \varnothing$
9: **if** p_1, p_2 are both doors **then** $dist \leftarrow v_k.\mathsf{M}_{d2d}(p_1, p_2)$
10: **else** $dist \leftarrow |p_1, p_2|_E$
11: get $\mathsf{P}^k_{t^a}$ from G
12: $time \leftarrow T(p_1, p_2, v_k, t^a)$ ▷ Equation D.3
13: **if** QT = LCPQ **then**
14: $contact \leftarrow \kappa(p_1, p_2, v_k, t^a)$ ▷ Equation D.4
15: **return** $(dist, time, contact)$
16: **else return** $(dist, time)$

is modified to $(v_j, v_i, d_k).\mathsf{F}[t_c] \leftarrow f(v_j, v_i, d_k).\lambda$. That is, the outflow of an adjacent partition v_j is directly obtained from the corresponding door flow function.

Strategy 2: Population Derivation at Necessary Timestamps (NT). To further speed up the population derivation for individual partitions, we consider reducing the workload by only calling Algorithm 16 at some necessary timestamps. The general idea is that when we observe that the historical flows of a partition are relatively stable, we skip the iterative population computations and directly estimate its population at the arrival time t^a. Note that here Strategy 2 is used in combination with Strategy 1 to achieve the maximum effect of acceleration.

In particular, when the search visits a partition v_k, the mean μ and standard deviation σ of its flow difference (i.e., inflow deducts outflow) in the historical timestamps are computed as follows.

$$\mu = \Big(\sum_{t_x \in \mathsf{UT}_{past}} \big(in(v_k, t_x) - out(v_k, t_x) \big) \Big) / |\mathsf{UT}_{past}|$$

$$\sigma = \Big(\big(\sum_{t_x \in \mathsf{UT}_{past}} (in(v_k, t_x) - out(v_k, t_x) - \mu)^2 \big) / |\mathsf{UT}_{past}| \Big)^{1/2}$$

where UT_{past} is a set of the historical update timestamps of v_k. The update timestamps in UT_{past} will be obtained from the local array that we maintain for fitting door flow function in Section D.3.2.

If σ is smaller than a pre-defined threshold value η, it indicates that v_k's historical flows change only slightly. Thus, we directly estimate v_k's population according to its

historical trend as follows.

$$\mathsf{P}^k_{t^a} = \mathsf{P}^k_{t_l} + \mu \cdot |\{t \in UT(v_k) \mid t_l < t \wedge t \leq t^a\}| \tag{D.7}$$

where $t_l = \max\{t_x \in UT_G \mid t_x \leq t_q\}$ is the latest population update time as in line 5 of Algorithm 17, μ is the mean of historical flow differences, and $|\{t \in UT(v_k) \mid t_l < t \wedge t \leq t^a\}|$ is the number of skipped update timestamps from t_l to t^a. In our experiment, $\eta = 3$ achieves the best performance approaching exact search results.

Otherwise, the search has to call Algorithm 16 (applied with Strategy 1) to derive population for v_k.

D.5.3 Complexity Analysis

The main difference of the four algorithms' complexity is related to population derivation. Therefore, we focus on comparing the four ways of population derivation. Assume that we estimate a partition's population at a future timestamp, and the derivation involves k unit time intervals.

The time complexity of the global population estimator (Algorithm 15) is $k|V| \cdot \mathtt{u}$, where $\mathtt{u}$ is the unit computational cost for a partition at an update timestamp. For the local estimator (Algorithm 16) which only considers the current partition and its upstream partitions, its time complexity is $\big(k|V| - ((k-1)n^k + (k-2)n^{k-1} + \cdots + n^2)\big) \cdot \mathtt{u}$, where n is the average number of enterable door per partition.

For two approximate strategies, PP rectifies only the outflows of the current certain partition without strictly processing the outflows of its upstream partitions, so the time complexity of PP's population derivation per partition is only $k\mathtt{u}$. NT omits population estimations for some partitions with the relatively stable flow. Thus for a partition in consideration, the time complexity depends on its flow stability. That is, if it is stable, we do not estimate the population; otherwise, the complexity is also $k\mathtt{u}$.

D.6 Experiments

For either FPQ or LCPQ, we implement four search algorithms. Specifically, *PQ is Algorithm 17 calling Algorithm 16, *PQ-G is Algorithm 17 calling Algorithm 15, *PQ-PP is the approximate search using Strategy PP, and *PQ-NT is the approximate search using Strategy NT. All algorithms are implemented in Java and run on a PC with a 2.30GHz Intel i5 CPU and 16 GB memory.

D.6.1 Results on Synthetic Data

Settings

Indoor Space. Using a real-world floorplan[4], we generate a multi-floor indoor space where each floor takes 1368m $\times$ 1368m. The irregular hallways are decomposed into smaller but regular partitions following the decomposition algorithm in [25]. As a result, we obtain 141 partitions and 216 doors on each floor. We duplicate the floorplan 3, **5**, 7, or 9 times to simulate different indoor spaces. All parameter settings are listed in Table D.2 with default values in bold. The four staircases of each two adjacent floors are connected by stairways, each being 20m long. On each floor, we randomly pick 14 out of all those partitions having two doors as the Q-partitions while regarding all others as R-partitions.

Table D.2: Parameter Settings

Parameters	Description	Settings
floor	Floor number	3, **5**, 7, 9
$\|o\|$	Partition's maximum object number	300, **600**, 900, 1200, 1500
TI (s)	Time interval	5, **10**, 15, 20
s2t (m)	The shortest distance from p_s to p_t	900, 1100, **1300**, 1500, 1700

Populations and Flows. We generate each partition's population at an initial time randomly from 0 to $|o|$ (see Table D.2). We set the max capacity of a partition v as $Area(v) \cdot \beta$ (β is 1 per m^2 in this paper). Note that the initial population will not exceed the max capacity. The parameter λ of each door flow function is varied from 0 to 3 [5]. We use a variable *TI* (5, **10**, 15, or 20 seconds) to control the length of the unit update time interval of partitions. To this end, all doors' initial report timestamps are aligned and they only report the flows in every $n \cdot TI$ seconds ($n = 1, 2, \ldots, 5$ is randomly decided for each door counter).

Query Instances. We use a parameter *s2t* to control the shortest distance from the source point p_s and the target point p_t. First, we randomly select a point p_s from the indoor space. Second, we find a door d whose indoor distance to p_s approximates *s2t*. Then, we expand from d to find a random point p_t whose indoor distance to p_s approaches *s2t*. For each *s2t* value, we generate 100 different pairs to form query instances.

Baseline Methods. We use a general time-dependent graph (GTG) to form a baseline. Each vertex in GTG represents a door and the weight of each edge is the cost between two doors, i.e., the time cost for `FPQ` or the contact for `LCPQ`. To be fair, we employ a Dijkstra-based algorithm (`*PQ-GTG`) without precomputation and combine it with our exact population estimator to process queries. Since GTG fails to represent the door directionality directly, we assume all doors are bidirectional in comparative experiments. Another baseline is the adaptive method based on the indoor

[4] `https://longaspire.github.io/s/fp.html` (Last accessed date: 2021/04/16)

[5] The value is set according to our analysis of real data. The door flow of a hallway/staircase is relatively more than that of a room.

Table D.3: Comparison of Algorithms for FPQ on Synthetic Data (best result in bold)

	FPQ	FPQ-G	FPQ-PP	FPQ-NT	FPQ-GTG	FPQ-A
Running Time (ms)	584	585	208	**25**	2857	189
Memory (KB)	115	112	111	**12**	278	14
Hit Rate (%)	**98**	**98**	**98**	95	**98**	94
Relative Error	**4.37E-08**	**4.37E-08**	**4.37E-08**	8.09E-08	**4.37E-08**	0.1233

crowd model (*PQ-A) that keeps updating and recomputing the optimal route at every node until the user gets to the target point.

Performance Metrics. To compare the efficiency of different search algorithms, we run each query instance ten times and measure their average running time and memory cost. We also look into the accuracy of the four searches. In particular, the query *hit rate* is the fraction of query instances whose search result equals its gold standard result among all 100 instances. The gold result is returned by searching over the detailed simulated trajectories. Moreover, we measure the *relative error* of the estimated routing cost against the gold result. The estimated cost refers to overall travel time T_ϕ for FPQ and overall contact κ_ϕ for LCPQ. Taking FPQ as an example, the relative error is $\gamma = |T_\phi^{(E)} - T_\phi^{(G)}|/T_\phi^{(G)}$ where $T_\phi^{(E)}$ and $T_\phi^{(G)}$ is the overall travel time corresponding to the exact search and gold result, respectively.

Search Performance of FPQ

Comparison in default setting. The measures of different FPQ algorithms are reported in Table D.3. FPQ-NT performs the best in terms of the running time and memory because it skips the iterative population computations and directly estimates its population at the arrival time in each node. FPQ and FPQ-G perform similarly as two exact searches, implying that the two exact estimators achieve similar efficiency in the default setting. Besides, they are the best in terms of hit rate and relative error. The baseline FPQ-GTG uses the exact estimator that we propose, so its accuracy is the same as FPQ and FPQ-G. However, FPQ-GTG incurs the highest time and memory costs due to the large size of GTG (cf. Section D.3). FPQ-PP works as accurately as the exact algorithms, which reflects the effectiveness of Strategy PP. Also, FPQ-PP saves some time and memory. FPQ-NT and FPQ-A perform worse in terms of hit rate and relative error. FPQ-NT skips some intermediate update timestamps, making its population derivation less accurate. FPQ-A expands to next nodes by reevaluation, making its result only optimal locally rather than globally. Note that the running time (and memory) of FPQ-A is the sum of that at all nodes in a path. Indeed, FPQ-A keeps updating until a user gets to the target point, while other methods return the path before departure. We omit FPQ-GTG and FPQ-A in the subsequent experiments as the comparison results show a similar trend to that here.

Effect of *s2t*. We vary the distance *s2t* between p_s and p_t from 900m to 1700m and test the four FPQ algorithms. Referring to Figure D.5, all algorithms' running time increases linearly with the source-target distance, since a larger *s2t* involves a

larger expansion range as well as more candidate path nodes. Among all algorithms, FPQ-NT runs fastest because it skips the iterative population computation and directly estimates its population at the arrival time in each node. Moreover, the time costs of approximate searches FPQ-PP and FPQ-NT increase very slowly as *s2t* increases. In contrast, the exact searches FPQ and FPQ-G are sensitive to *s2t* because they need to compute more population.

Figure D.6 reports on the memory consumption. The memory use of FPQ and FPQ-G grows faster than the others due to an extra cost of rigid population derivation. Contrary to our intuition, FPQ incurs more memory cost than FPQ-G. In our test, the search framework needs to explore a large number of partitions. FPQ-G's global population derivation shares intermediate results across all partitions. For a large *s2t*, FPQ-G may find more shared intermediate results, and thus consumes less memory than FPQ.

Figure D.7 reports on the relative errors, for FPQ, FPQ-PP and FPQ-NT.[6] For different *s2t* values, FPQ achieves a lower error. As a sacrifice for search efficiency, FPQ-NT skips some intermediate update timestamps, so its accuracy of population derivation drops more significantly. As *s2t* increases, FPQ-NT deteriorates rapidly while FPQ and FPQ-PP perform quite stably. A larger *s2t* leads to more updated timestamps to derive populations. As a result, FPQ-NT's relatively aggressive strategy of skipping timestamps incurs more estimation errors. In contrast, FPQ and FPQ-PP derive populations timestamp by timestamp, and so the impact of *s2t* is slight.

Effect of *TI*. According to Figure D.8, all four algorithms run faster with a larger update time interval *TI*. Still, FPQ-NT performs best in both measures as it approximates population derivation in both time and space aspects. We omit the result of the memory cost because it has a similar trend with the running time. On the other hand, referring to Figure D.9, FPQ-NT's relative error decreases as all doors' *TI* enlarges. This shows that one may consider skipping more timestamps when the flow update at doors is not that frequent.

Effect of *floor*. We vary the floor number to test the scalability of our algorithms. Referring to Figure D.10, all algorithms' search time increases steadily with more floors since more candidate path nodes are involved. FPQ-PP and FPQ-NT run faster than FPQ and FPQ-G. Moreover, the running time of the two approximate searches grows more slowly. Figure D.11 reports the memory cost of the four algorithms. FPQ-NT needs less memory and is more scalable since it skips some timestamps. We omit the results of relative errors. In the tests, both measures are insensitive to the floor number since the returned path stays unchanged for a given query instance.

We omit the results of varying $|o|$ on FPQ because different initial object numbers have little impact on the search performance.

[6] We exclude FPQ-G as its accuracy is the same with FPQ.

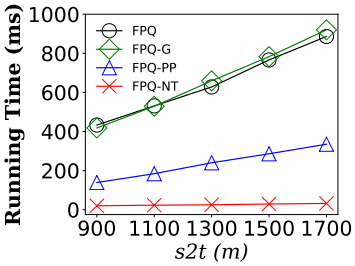


Fig. D.5: FPQ Time vs. *s2t*

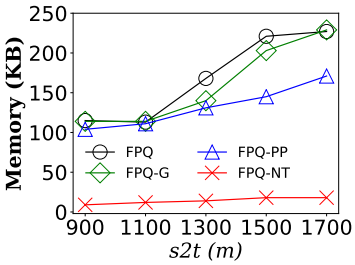


Fig. D.6: FPQ Mem. vs. *s2t*

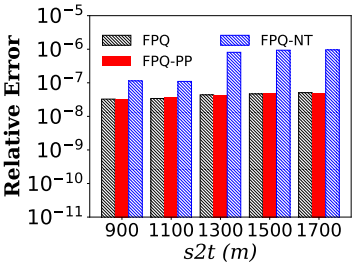


Fig. D.7: FPQ's γ vs. *s2t*

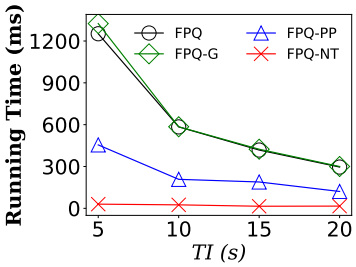


Fig. D.8: FPQ Time vs. *TI*

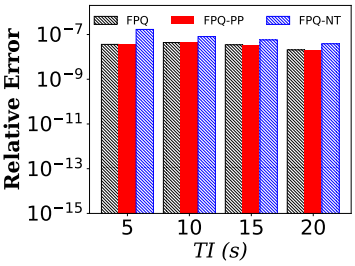


Fig. D.9: FPQ's γ vs. *TI*

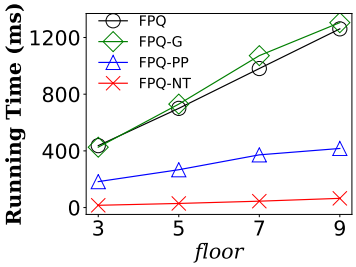


Fig. D.10: FPQ Time vs. *floor*

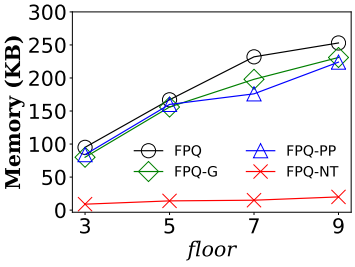


Fig. D.11: FPQ Mem. vs. *floor*

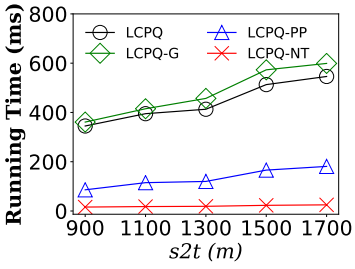


Fig. D.12: LCPQ Time vs. *s2t*

Search Performance of LCPQ

Comparison in default setting. The resulting trend of LCPQ is similar to that of FPQ. As reported in the right part of Table D.4, LCPQ-NT is the best in terms of running time and memory due to its skipping strategy, while LCPQ-GTG incurs the highest time

and memory costs due to the large graph size. LCPQ-A gets the best hit rate while the exact searches achieve a better result on the relative error. Different from FPQ, LCPQ is highly sensitive to populations. A little error in population derivation can lead to a very different returned path. Hence, the accuracy performance is slightly unstable for the tested algorithms.

Table D.4: Comparison of Algorithms for LCPQ on Synthetic Data (best result in bold)

	LCPQ	LCPQ-G	LCPQ-PP	LCPQ-NT	LCPQ-GTG	LCPQ-A
Running Time (ms)	446	461	131	**20**	2532	163
Memory (KB)	182	192	144	**7**	257	8
Hit Rate (%)	83	83	83	60	83	**87**
Relative Error	**0.0128**	**0.0128**	0.0129	0.1113	**0.0128**	0.1256

Effect of *s2t*. Referring to Figure D.12, the running time of each algorithm grows as *s2t* increases. In terms of memory, the results in Figure D.13 show that LCPQ and LCPQ-G need more memory than the other two. LCPQ-NT uses the least memory since it skips intermediate timestamps to reduce workload.

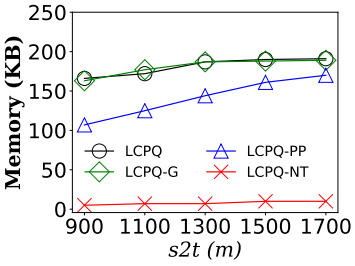


Fig. D.13: LCPQ Mem. vs. *s2t*

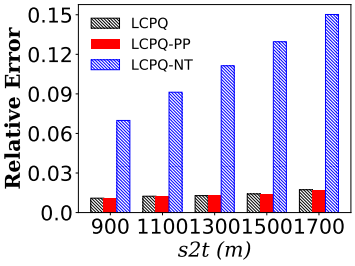


Fig. D.14: LCPQ's γ vs. *s2t*

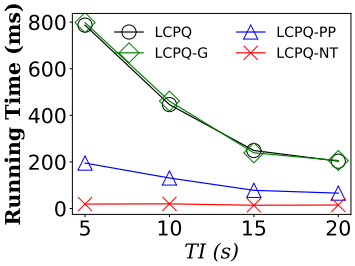


Fig. D.15: LCPQ Time vs. *TI*

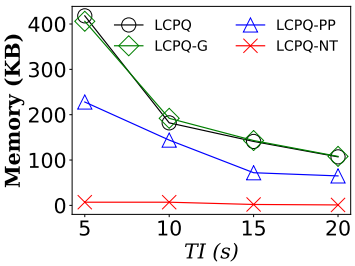


Fig. D.16: LCPQ Mem. vs. *TI*

Figure D.14 reports on the relative errors of exact and approximate searches. Compared to LCPQ and LCPQ-PP, LCPQ-NT incurs significantly higher errors. As we mentioned before, LCPQ query is highly sensitive to the population. A little error in population derivation can lead to a very different returned path. Therefore, LCPQ-NT performs poorly when a larger *s2t* is used.

Effect of *TI*. Referring to Figures D.15 and D.16, all algorithms incur less time and memory costs as *TI* increases since a larger *TI* leads to fewer callings of population derivation. The approximate approaches LCPQ-PP and LCPQ-NT always perform

better in search efficiency. Referring to Figure D.17, all algorithms' search effectiveness deteriorates with an increasing *TI*. As less flow information is observed when *TI* becomes larger, the relative errors accumulate. Likewise, LCPQ's search effectiveness is worse than that of FPQ, due to its more stringent requirements on population derivation.

Effect of $|o|$. We test different initial object numbers on LCPQ query processing. As an observation, the running time and memory cost are almost insensitive to $|o|$, since the algorithms do not process each individual object. So we omit the results here. Interestingly, increasing $|o|$ will affect the result accuracy. Referring to Figure D.18, as more objects are involved, all methods achieve a lower relative error. We attribute it to that a larger population base is less affected by the flow estimation error and leads to a smaller relative error.

We omit the result about different floor numbers because it exhibits a trend similar to the counterpart of FPQ searches.

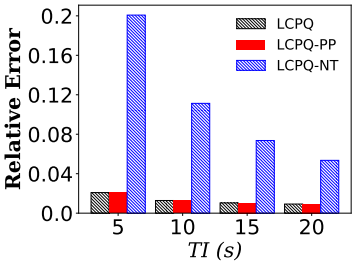


Fig. D.17: LCPQ's γ vs. *TI*

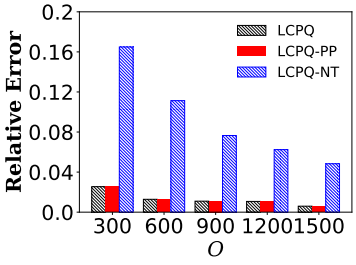


Fig. D.18: LCPQ's γ vs. *O*

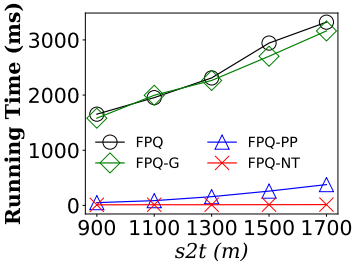


Fig. D.19: FPQ Time vs. *s2t* (Real)

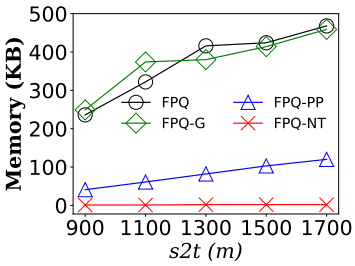


Fig. D.20: FPQ Mem. vs. *s2t* (Real)

D.6.2 Results on Real Data

We collect a real dataset from a seven-floor, 2700m $\times$ 2000m shopping mall in Hangz hou, China. There are ten staircases each being roughly 20m long, and 977 partitions connected by 1613 doors[7]. The max capacity of a partition v is $Area(v) \cdot$

[7]We assume that there is no Q-partition in this shopping mall. We varied the fraction of Q-partitions/R-partitions on synthetic data, but it shows little impact on all algorithms.

1 per m^2. We collected 1,598 object trajectories with totally more than 90,000 positioning records on 2017/01/05. Nearly 12% of two consecutive locations are not topologically-connected, i.e., not in the same partition or two adjacent partitions. The object movements in-between are uncertain. To count flows against uncertainty, we applied a proven probabilistic method [16] as follows. First, for every two consecutive locations not topologically-connected, a set Φ of valid sub-paths are found. Those sub-paths longer than twice the shortest sub-path are excluded as the object unlikely took them. Second, the probability that the object took sub-path $\phi_i \in \Phi$ is computed as $\mathsf{P}(\phi_i) = \frac{1/length(\phi_i)}{\sum_{\phi_k \in \Phi} 1/length(\phi_k)}$. This way, a shorter sub-path has a higher probability to be taken. Finally, the flow of a door d is the sum of $\mathsf{P}(\phi_i)$s for all ϕ_is through d. On top of the low-level flow computing, we sampled each door's flow every 10 seconds and used the samples to construct our indoor crowd model.

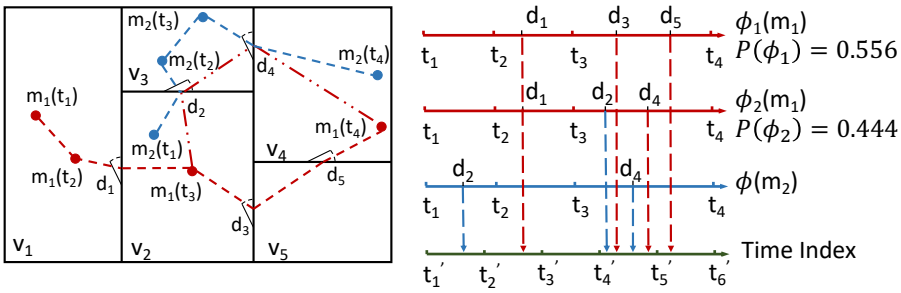


Fig. D.21: An Example of the Trajectory Data

Figure D.21 exemplifies a few trajectories, where $m_i(t_j)$ denotes the positioning location of a MAC address m_i at time t_j. For $m_1(t_3)$ and $m_1(t_4)$ that are not topologically-connected, two possible in-between paths are found, namely $\phi_1(m_1(t_3), d_3, d_5), m_1(t_4))$ of 20m long and $\phi_2(m_1(t_3), d_2, d_4), m_1(t_4))$ of 25m long. Their probabilities are $\mathsf{P}(\phi_1) = \frac{1/20}{1/20+1/25} \approx 0.556$ and $\mathsf{P}(\phi_2) = \frac{1/25}{1/20+1/25} \approx 0.444$. We sampled door flows as shown in the right part of Figure D.21. E.g., door d_4's flow during $[t_4', t_5']$ is $1 + 0.444 = 1.444$ (m_2 with a probability of 1 and m_1 with a probability of 0.444).

Comparison in default setting. We compare different methods for `FPQ` and `LCPQ` using real data and report the results in Tables D.5 and D.6. In terms of the running time and memory, `*PQ-NT` performs best while `*PQ-GTG` is the worst. This is because `*PQ-NT` skips the iterative population computations while `*PQ-GTG` uses an exact estimator but involves more nodes than does our indoor crowd model. In terms of hit rate and relative error, the results are similar to the counterparts in synthetic data.

Effect of *s2t*. Figures D.19 and D.20 report on running time and memory use of different `FPQ` searches. All incur more time and memory costs as *s2t* increases. Compared to `FPQ-PP` and `FPQ-NT`, `FPQ` and `FPQ-G` are more memory- and time-consuming. Compared to the counterparts on synthetic data, the search costs are higher since the shopping mall is of a larger scale. Referring to Figure D.22, `FPQ-PP`/`FPQ-NT`'s search

Table D.5: Comparison of Algorithms for FPQ on Real Data (best result in bold)

	FPQ	FPQ-G	FPQ-PP	FPQ-NT	FPQ-GTG	FPQ-A
Running Time (ms)	1900	1997	67	**11**	25559	53
Memory (KB)	367	393	61	**1**	669	2
Hit Rate (%)	**99**	**99**	**99**	98	**99**	98
Relative Error	**1.86E-15**	**1.86E-15**	**1.86E-15**	4.38E-14	**1.86E-15**	0.1492

Table D.6: Comparison of Algorithms for LCPQ on Real Data (best result in bold)

	LCPQ	LCPQ-G	LCPQ-PP	LCPQ-NT	LCPQ-GTG	LCPQ-A
Running Time (ms)	992	1047	28	**10**	13895	45
Memory (KB)	307	341	30	**1**	568	2
Hit Rate (%)	88	88	88	67	88	**90**
Relative Error	**0.0546**	**0.0546**	**0.0546**	0.6606	**0.0546**	0.062

accuracy deteriorates with a larger *s2t*.

The results in Figures D.23 and D.24 exhibit similar trends as those in Figures D.19 and D.20. LCPQ-NT's relative error is much higher than that of LCPQ and LCPQ-PP, and it grows faster—more update timestamps involved due to a greater *s2t* lead to less accurate cost estimates. Compared to FPQ, LCPQ has higher relative errors as reported in Figure D.25. As defined, partition-passing contact is more sensitive to the derived populations than the partition-passing time.

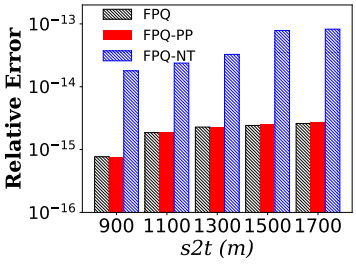


Fig. D.22: FPQ's γ vs. *s2t* (Real)

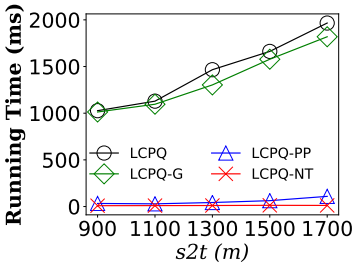


Fig. D.23: LCPQ Time vs. *s2t* (Real)

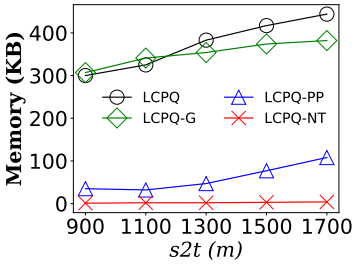


Fig. D.24: LCPQ Mem. vs. *s2t* (Real)

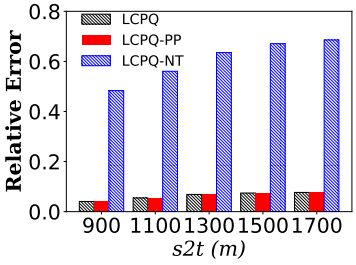


Fig. D.25: LCPQ's γ vs. *s2t* (Real)

D.6.3 Summary of Results

First, in terms of running time and memory, the two approximate searches perform better than the two exact counterparts as workloads reduce. Besides, Strategy NT costs less time and memory than Strategy PP since NT further utilizes historical information to skip timestamps. For hit rate and relative error, PP outperforms NT in that NT skips many timestamps on the basis of PP, further decreasing the accuracy of intermediate results.

Second, the two approximate searches for FPQ perform better than those for LCPQ in terms of hit rate and error rate. The reason is that the partition-passing time is less sensitive to the populations compared to the partition-passing contact.

Third, a larger *s2t* leads to more time and memory consumptions but worse result accuracy, while a larger *TI* exhibits an almost opposite trend. In general, a larger *s2t* or a smaller *TI* means more timestamps for which we need to derive populations. This is critical to the cost estimation. More floors mean more doors/partitions to explore, which lowers the search efficiency. More initial objects render the population derivation spatially more uniform and thus leads to a higher hit rate and lower relative error.

Fourth, for the two baselines, *PQ-GTG performs poorly on efficiency because GTG contains more nodes to process. *PQ-A seems good in terms of both efficiency and effectiveness. However, a user of *PQ-A cannot obtain the path before departure because *PQ-A needs to keep updating during expansion.

In general, the results show that the search algorithm with Strategy PP performs best. It costs relatively less time and memory and achieves good query result accuracy. Strategy NT applies well to the cases where door flows are updated frequently. In such a case, skipping some timestamps can improve efficiency without causing excessive errors in population estimates. More experimental results are available in an extended version [26].

D.7 Related Work

Outdoor Time-Dependent Routing. In this setting, public transportation networks [7–9] and road networks [10–14] are modeled as discrete and continuous time-dependent graphs, respectively. Solutions for public transportation networks cannot solve our problem because they are mainly for a time-dependent graph with a static timetable for each station. On the other hand, approaches for road networks do not work for indoor spaces because road network models do not support entities like doors, walls and rooms that together form a complex topology.

From an algorithmic perspective, the solutions for outdoor time-dependent routing are mainly Dijkstra-based algorithms [8, 10, 20, 27], A* algorithms [12, 13], label-based methods [9, 28–30] (mainly for time-dependent graphs with timetables), and adaptive approaches [11, 13, 31]. Most of these works do not consider crowds that

influence people's routing choices.

Traffic-aware Routing. Some works [12, 32, 33] prepare the traffic information by mining historical trajectory data and assume it is known when routing. Shang et al. [34] study traffic-aware fastest path query using a traffic-aware spatial network. Some adaptive approaches [11, 13, 31] can also solve traffic-aware routing problems through continuous reevaluation. Although these works consider the traffic impact, none of them estimates the traffic in the near future when processing a query.

Indoor Routing. Lu et al. [1] propose a distance-aware indoor space model to facilitate indoor shortest path query. To speed up distance-aware indoor pathfinding, Shao et al. [2] design IP/VIP-tree that enable more aggressive pruning. VIP-tree also supports trip planning based on neighbour expansion [15]. Feng et al. [6] study indoor top-k keyword-aware routing query that finds k routes that have optimal ranking scores integrating keyword relevance and distance cost. Other works [3, 4, 35–37] consider more constraints for indoor routing. However, none of these aforementioned works considers dynamic crowds that are essential to `LCPQ` and `FPQ`.

Flows, Crowds and Density. Some existing works study the estimation of flows [38, 39], crowds [40, 41], or dense regions [42, 43] outdoors. However, they all fall short in indoor spaces mainly for two reasons. First, indoor positioning techniques are usually RFID, Wi-Fi, and Bluetooth, which make coarser-grained location data than outdoor GPS data. Second, the indoor topology is so different from the outdoor topology that indoor crowd modeling must consider carefully the connectivity among doors and partitions. Some existing works consider flows and densities in indoor venues. Ahmed et al. [44] propose two graph-based indoor movement models to map raw tracking records into records with object entry and exit times in particular locations. Li et al. propose to find the top-k popular indoor semantic locations [16] with the highest flow values using probabilistic location samples, and the currently top-k indoor dense regions [17] by considering the uncertainty of online positioning reports. However, all these works [16, 17, 44] are different from our work. First, our density analysis is based on coarse-grained flow values reported at door counters, not the point-based localization results count for individual moving objects. Second, our work focuses on path planning in the presence of indoor crowds, while the previous works aim to find interesting location patterns.

D.8 Conclusion and Future Work

We study two types of crowd-aware indoor path planning queries. The `FPQ` returns a path with the shortest travel time in the presence of crowds; the `LCPQ` returns a path encountering the least objects en route. To solve `FPQ` and `LCPQ`, we design a unified framework that consists of 1) an indoor crowd model that organizes indoor topology and captures indoor flows and densities; 2) a time-evolving population estimator that derives future time-dependent flows and populations for relevant partitions; 3) two exact and two approximate query processing algorithms that each can process both

query types. We conduct extensive experiments to evaluate our proposals. The results demonstrate the efficiency and scalability of the proposals and disclose the performance differences among all four algorithms.

There exist several directions for future research. First, it is interesting to consider other crowd models, e.g., learning crowd distributions and functions from historical data. Also, it is relevant to further speed up query processing by using an index, e.g., combining the object layer in the composite indoor index [25, 45] with a modified IP/VIP-Tree [2] whose distance matrices are extended with time attributes. Last but not least, it is possible to extend our proposals to support continuous monitoring of the fastest or least crowded paths.

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). Huan Li and Hua Lu are the corresponding authors.

References

[1] H. Lu, X. Cao, and C. S. Jensen, "A foundation for efficient indoor distance-aware query processing," in *2012 IEEE 28th International Conference on Data Engineering*. IEEE, 2012, pp. 438–449.

[2] Z. Shao, M. A. Cheema, D. Taniar, and H. Lu, "Vip-tree: an effective index for indoor spatial queries," *Proceedings of the VLDB Endowment*, vol. 10, no. 4, pp. 325–336, 2016.

[3] W. Luo, P. Jin, and L. Yue, "Time-constrained sequenced route query in indoor spaces," in *Asia-Pacific Web Conference*. Springer, 2016, pp. 129–140.

[4] Y. Zhou, H. Chen, Y. Huang, Y. Luo, Y. Zhang, and X. Xie, "An indoor route planning method with environment awareness," in *IGARSS 2018-2018 IEEE International Geoscience and Remote Sensing Symposium*. IEEE, 2018, pp. 2906–2909.

[5] H. Li, H. Lu, F. Shi, G. Chen, K. Chen, and L. Shou, "Trips: A system for translating raw indoor positioning data into visual mobility semantics," *Proceedings of the VLDB Endowment*, vol. 11, no. 12, pp. 1918–1921, 2018.

[6] Z. Feng, T. Liu, H. Li, H. Lu, L. Shou, and J. Xu, "Indoor top-k keyword-aware routing query," in *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 2020, pp. 1213–1224.

[7] R. W. Hall, "The fastest path through a network with random time-dependent travel times," *Transp. Sci.*, vol. 20, no. 3, pp. 182–188, 1986.

[8] G. S. Brodal and R. Jacob, "Time-dependent networks as models to achieve fast exact time-table queries," *ENTCS*, vol. 92, pp. 3–15, 2004.

[9] S. Wang, W. Lin, Y. Yang, X. Xiao, and S. Zhou, "Efficient route planning on public transportation networks: A labelling approach," in *SIGMOD*, 2015, pp. 967–982.

[10] B. Ding, J. X. Yu, and L. Qin, "Finding time-dependent shortest paths over large graphs," in *Proceedings of the 11th international conference on Extending database technology: Advances in database technology*, 2008, pp. 205–216.

[11] M. K. Ardakani and L. Sun, "Decremental algorithm for adaptive routing incorporating traveler information," *Computers & operations research*, vol. 39, no. 12, pp. 3012–3020, 2012.

[12] G. Nannicini, D. Delling, D. Schultes, and L. Liberti, "Bidirectional A* search on time-dependent road networks," *Networks*, vol. 59, no. 2, pp. 240–251, 2012.

[13] M. K. Ardakani and M. Tavana, "A decremental approach with the A* algorithm for speeding-up the optimization process in dynamic shortest path problems," *Measurement*, vol. 60, pp. 299–307, 2015.

[14] V. J. Wei, R. C.-W. Wong, and C. Long, "Architecture-intact oracle for fastest path and time queries on dynamic spatial networks," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, 2020, pp. 1841–1856.

[15] Z. Shao, M. A. Cheema, and D. Taniar, "Trip planning queries in indoor venues," *The Computer Journal*, vol. 61, no. 3, pp. 409–426, 2018.

[16] H. Li, H. Lu, L. Shou, G. Chen, and K. Chen, "Finding most popular indoor semantic locations using uncertain mobility data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 11, pp. 2108–2123, 2018.

[17] ——, "In search of indoor dense regions: An approach using indoor positioning data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 8, pp. 1481–1495, 2018.

[18] M. VIRKLER and S. ELAYADATH, "Pedestrian speed-flow-density relationships: Pedestrians and pedestrian facilities," *Transportation research record*, no. 1438, pp. 51–58, 1994.

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

[20] Y. Yuan, X. Lian, G. Wang, Y. Ma, and Y. Wang, "Constrained shortest path query in a large time-dependent graph," *PVLDB*, vol. 12, no. 10, pp. 1058–1070, 2019.

[21] J. Contreras, R. Espinola, F. J. Nogales, and A. J. Conejo, "ARIMA models to predict next-day electricity prices," *T-PWRS*, vol. 18, no. 3, pp. 1014–1020, 2003.

[22] Y. Liang, S. Ke, J. Zhang, X. Yi, and Y. Zheng, "Geoman: Multi-level attention networks for geo-sensory time series prediction," in *IJCAI*, 2018, pp. 3428–3434.

[23] P. C. Consul and G. C. Jain, "A generalization of the poisson distribution," *Technometrics*, vol. 15, no. 4, pp. 791–799, 1973.

[24] M. T. Boswell *et al.*, "Estimating and testing trend in a stochastic process of poisson type," *AMS*, vol. 37, no. 6, pp. 1564–1573, 1966.

[25] X. Xie, H. Lu, and T. B. Pedersen, "Distance-aware join for indoor moving objects," *IEEE Transactions on Knowledge and Data Engineering*, vol. 2, no. 27, pp. 428–442, 2015.

[26] T. Liu, H. Li, H. Lu, M. A. Cheema, and L. Shou, "Towards crowd-aware indoor path planning (extended version)," *arXiv preprint arXiv:2104.05480*, 2021.

[27] K. L. Cooke and E. Halsey, "The shortest route through a network with time-dependent internodal transit times," *JMAA*, vol. 14, no. 3, pp. 493–498, 1966.

[28] K. Nachtigall, "Time depending shortest-path problems with applications to railway networks," *EJOR*, vol. 83, no. 1, pp. 154–166, 1995.

[29] H. Wu, J. Cheng, S. Huang, Y. Ke, Y. Lu, and Y. Xu, "Path problems in temporal graphs," *PVLDB*, vol. 7, no. 9, pp. 721–732, 2014.

[30] H. Wu, Y. Huang, J. Cheng, J. Li, and Y. Ke, "Reachability and time-based path queries in temporal graphs," in *ICDE*, 2016, pp. 145–156.

[31] H. Gonzalez, J. Han, X. Li, M. Myslinska, and J. P. Sondag, "Adaptive fastest path computation on a road network: a traffic mining approach," in *VLDB*, 2007, pp. 794–805.

[32] U. Demiryurek, F. Banaei-Kashani, C. Shahabi, and A. Ranganathan, "Online computation of fastest path in time-dependent spatial networks," in *SSTD*, 2011, pp. 92–111.

[33] Y. Wang, G. Li, and N. Tang, "Querying shortest paths on time dependent road networks," *PVLDB*, vol. 12, no. 11, pp. 1249–1261, 2019.

[34] S. Shang, H. Lu, T. B. Pedersen, and X. Xie, "Finding traffic-aware fastest paths in spatial networks," in *SSTD*, 2013, pp. 128–145.

[35] L. Liu, S. Zlatanova, B. Li, P. van Oosterom, H. Liu, and J. Barton, "Indoor routing on logical network using space semantics," *ISPRS INT J GEO-INF*, vol. 8, no. 3, p. 126, 2019.

[36] D.-H. Kim, B. Jang, and J. W. Kim, "Privacy-preserving top-k route computation in indoor environments," *IEEE Access*, vol. 6, pp. 56 109–56 121, 2018.

[37] T. Liu, Z. Feng, H. Li, H. Lu, M. A. Cheema, H. Cheng, and J. Xu, "Shortest path queries for indoor venues with temporal variations," in *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 2020, pp. 2014–2017.

[38] Y. Tao, G. Kollios, J. Considine, F. Li, and D. Papadias, "Spatio-temporal aggregation using sketches," in *ICDE*, 2004, pp. 214–225.

[39] C. Tang, J. Sun, Y. Sun, M. Peng, and N. Gan, "A general traffic flow prediction approach based on spatial-temporal graph attention," *IEEE Access*, vol. 8, pp. 153 731–153 741, 2020.

[40] G. Castellano, C. Castiello, C. Mencar, and G. Vessio, "Crowd detection in aerial images using spatial graphs and fully-convolutional neural networks," *IEEE Access*, vol. 8, pp. 64 534–64 544, 2020.

[41] S. Wang, Y. Lu, T. Zhou, H. Di, L. Lu, and L. Zhang, "SCLNet: Spatial context learning network for congested crowd counting," *Neurocomputing*, vol. 404, pp. 227–239, 2020.

[42] X. Huang and H. Lu, "Snapshot density queries on location sensors," in *MobiDE*, 2007, pp. 75–78.

[43] X. Hao, X. Meng, and J. Xu, "Continuous density queries for moving objects," in *MobiDE*, 2008, pp. 1–7.

[44] T. Ahmed, T. B. Pedersen, and H. Lu, "Finding dense locations in symbolic indoor tracking data: modeling, indexing, and processing," *GeoInformatica*, vol. 21, no. 1, pp. 119–150, 2017.

[45] X. Xie, H. Lu, and T. B. Pedersen, "Efficient distance-aware query evaluation on indoor moving objects," in *2013 IEEE 29th International Conference on Data Engineering (ICDE)*. IEEE, 2013, pp. 434–445.

ISSN (online): 2446-1628
ISBN (online): 978-87-7573-938-7

AALBORG UNIVERSITY PRESS