Towards Indoor Temporal-variation aware Shortest Path Query

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

Published in:
IEEE Transactions on Knowledge and Data Engineering

DOI (link to publication from Publisher):
10.1109/TKDE.2021.3076144

Publication date:
2023

Document Version
Accepted author manuscript, peer reviewed version

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2021.3076144, IEEE Transactions on Knowledge and Data Engineering

Towards Indoor Temporal-variation aware Shortest Path Query

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

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.

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 [8], [10], [16], [17], [25], [28].

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 [19], [26], [29] 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 [26] 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 [19] precomputes and stores the distances between all pairs of doors in an indoor venue. The state-of-the-art technique, IP-Tree/VIP-Tree [26], 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 [19], [25] 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

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2021.3076144, IEEE Transactions on Knowledge and Data Engineering

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 2).
- We present IT-GRAPH that effectively captures temporal changes and semantic properties of indoor venues (Section 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 4).
- We conduct extensive experiments on both real and synthetic data (Section 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 6 and conclude the paper and discuss future directions in Section 7.

In contrast to our preliminary work [18], 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 2.3). Second, it presents more technical details with a concrete example of the IT-GRAPH based approaches (Section 3.2). Third, it proposes the new index IT-INDEX (Section 4.1), an efficient index based query processing algorithm (Section 4.2), and a complexity analysis of all algorithms (Section 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 5).

2 PRELIMINARIES

Table 1 lists the frequently used notations in this paper.

TABLE 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
$AD_{\sqcupdot}(N)$	enterable access doors in a tree node N
$AD_{\sqcdot}(N)$	leavable access doors in a tree node N

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 1. *Referring to the floorplan of an office building in Figure 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.*

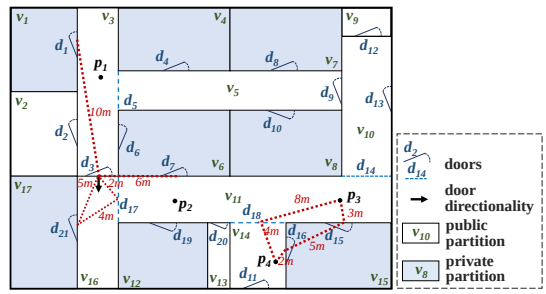


Fig. 1: An Example of Indoor FloorPlan

For ease of presentation, we show only one floor in Figure 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.

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 1 may be open and closed at different times as listed in Table 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 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 2. *In Table 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.*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2021.3076144, IEEE Transactions on Knowledge and Data Engineering

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.

Research Problem (Indoor Temporal-variation aware Shortest Path Query (ITSPQ))**.** *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 [1] — 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 3. *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.

2.3 Indoor Shortest Distance/Path Query Techniques

Indoor distance computation and path querying have been studied in the literature [19], [26].

Indoor Distance-Aware Model [19] 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.

VIP-Tree [26] 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 [19].

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.

3 ITSPQ USING TEMPORAL-VARIATION GRAPH

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

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 2.2) of the door.

The IT-GRAPH corresponding to Figure 1 is depicted in Figure 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 Figure 2, a record (d_1, *PRD*, ⟨[5:00, 23:00)⟩) 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.

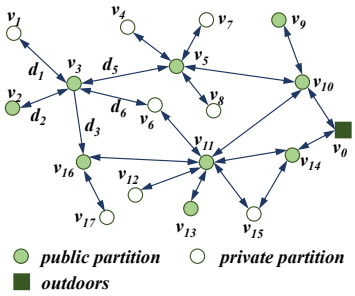
In general, IT-GRAPH combines indoor topology (i.e., graph structure), semantic properties of indoor entities (i.e., *d-type* and *p-type*), geometric information (i.e., *DM*), and temporal variation information (i.e., *ATIs*) in a composite structure.

Following the previous work [19], we also use several mapping functions to facilitate searching between partitions and doors.

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.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2021.3076144, IEEE Transactions on Knowledge and Data Engineering

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 Figure 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	⟨[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. 2: Example of Indoor Temporal-Variation Graph

3.2 IT-GRAPH based ITSPQ Processing

The overall framework for processing ITSPQ based on IT-GRAPH is presented in Algorithm 1. 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 $\texttt{TV\_Check}(d_j, DM(v, d_i, d_j), t_c)$ to check if d_j is open at the arrival time relative to the current time t_c (line 30). Two different strategies, namely $\texttt{Syn\_Check}()$ (Algorithm 2) and $\texttt{Asyn\_Check}()$ (Algorithm 4) 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).

Algorithm 1 $\texttt{ITSPQ\_ITGraph}(p_s, p_t, t, G_{IT})$

Input: Start point p_s, taget point p_t, query time t, and IT-GRAPH G_{IT}
Output: 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: $\quad dist[d_i] \leftarrow \infty$
4: $\quad$ enheap(H, $\langle d_i, dist[d_i] \rangle$)
5: $\quad prev[d_i] \leftarrow null$
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: $\quad \langle d_i, dist[d_i] \rangle \leftarrow$ deheap(H)
10: $\quad$ **if** $dist[d_i] = \infty$ **then return** no such routes
11: $\quad$ **if** $d_i = p_t$ **then**
12: $\quad\quad path \leftarrow p_t$
13: $\quad\quad$ **while** $prev[d_i] \neq p_s$ **do**
14: $\quad\quad\quad path \leftarrow prev[d_i] + ",\," + path$
15: $\quad\quad\quad d_i \leftarrow prev[d_i]$
16: $\quad\quad path \leftarrow p_s + ",\," + path$
17: $\quad\quad$ **return** $path$
18: $\quad$ **if** $d_i = p_s$ **then** $v \leftarrow \mathsf{P}(p_s)$ **else** $v \leftarrow D2P_{\sqsupset}(d_i) \backslash$ visited partitions
19: $\quad$ mark d_i and v as visited
20: $\quad$ **if** $d_i \in P2D_{\sqsupset}(\mathsf{P}(p_t))$ **then**
21: $\quad\quad$ **if** $dist[d_i] + |d_i, p_t|_E < dist[p_t]$ **then**
22: $\quad\quad\quad dist[p_t] \leftarrow dist[d_i] + |d_i, p_t|_E$
23: $\quad\quad\quad$ enheap(H, $\langle p_t, dist[p_t] \rangle$)
24: $\quad\quad\quad prev[p_t] \leftarrow (v, d_i)$
25: $\quad$ **else**
26: $\quad\quad$ **for** each unvisited door $d_j \in P2D_{\sqsubset}(v)$ **do**
27: $\quad\quad\quad v' \leftarrow D2P_{\sqsupset}(d_j) \backslash v$
28: $\quad\quad\quad$ **if** $v'.d\text{-}type$ is PRP **then** continue
29: $\quad\quad\quad dist_j \leftarrow dist[d_i] + DM(v, d_i, d_j)$; $t_c \leftarrow t + dist[d_i]/velocity$
30: $\quad\quad\quad$ **if** $!\texttt{TV\_Check}(d_j, DM(v, d_i, d_j), t_c)$ **then** continue
31: $\quad\quad\quad$ **if** $dist_j < dist[d_j]$ **then**
32: $\quad\quad\quad\quad dist[d_j] \leftarrow dist_j$
33: $\quad\quad\quad\quad$ enheap(H, $\langle d_j, dist[d_j] \rangle$)
34: $\quad\quad\quad\quad prev[d_j] \leftarrow (v, d_i)$

Example 4. *Corresponding to the ATIs information in Table 2, we want to find the shortest path from p_1 to p_2 at 11:00 in Figure 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 2, 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.

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

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2021.3076144, IEEE Transactions on Knowledge and Data Engineering

Algorithm 2 `Syn_Check`(d, $dist$, t_c)

Input: A door d, the distance $dist$, and current time t_c
Output: 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*

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 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 Figure 2. In contrast, when time goes between 16:00 and 17:00, the topology will be changed to the one illustrated in Figure 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

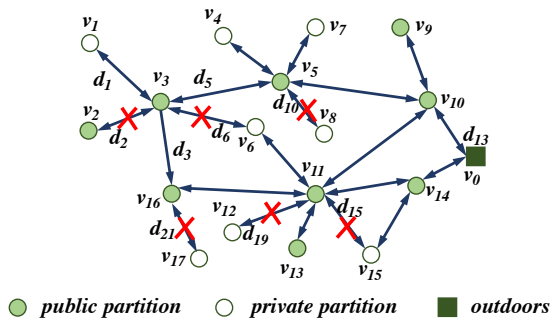


Fig. 3: Indoor Temporal-Variation Graph within [16:00,17:00)

refer to a 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 3. 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 1.

Algorithm 3 `Graph_Update`(t_{arr}, T)

Input: Arrival time t_{arr} and checkpoints set T
Output: 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 3, we present the asynchronous check in Algorithm 4. It first gets the current G_{IT} and its corresponding checkpoint cp (see line 6 in Algorithm 3) 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 3 (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 2 (line 6). It ensures that the expansion in lines 31-34 of Algorithm 1 will be executed.

Algorithm 4 `Asyn_Check`(d, $dist$, t_c)

Input: A door d, the distance $dist$, and current time t_c
Output: 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 1), 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 5.1.4.

4 ITSPQ USING TEMPORAL-VARIATION INDEX

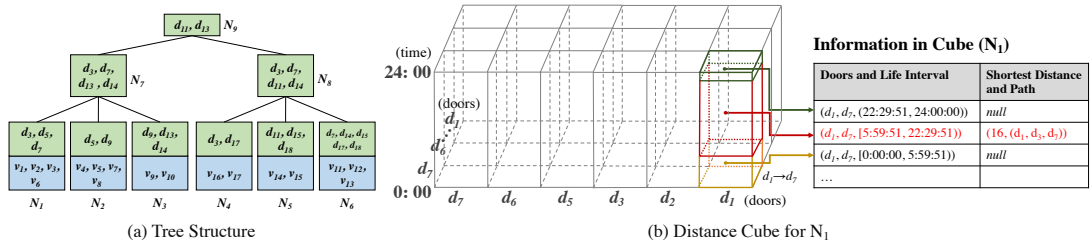
In Section 4.1, we present the **indoor temporal-variation index** (IT-INDEX) that organizes indoor partitions into a tree structure based on indoor topology. The indoor topology here refers to physical layout only and does not involve temporal variations and directionality of doors. Subsequently, we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

4.1 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 Figure 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 Figure 1, the tree structure of IT-INDEX is illustrated in Figure 4(a).

In IT-INDEX, each leaf node N_i maintains a set of *access doors* [26]. Based on door directionality, we distinguish **enterable access doors** and **leavable access doors** for N_i, the doors through



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

Fig. 4: Indoor Temporal-Variation Index

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 [26]. 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 [26].

Example 5. *Referring to the tree structure in Figure 4(a) that corresponds to the example in Figure 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 Figure 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 [26], 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 5 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 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 1, 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).

4.2 IT-INDEX based ITSPQ Processing

Figure 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.

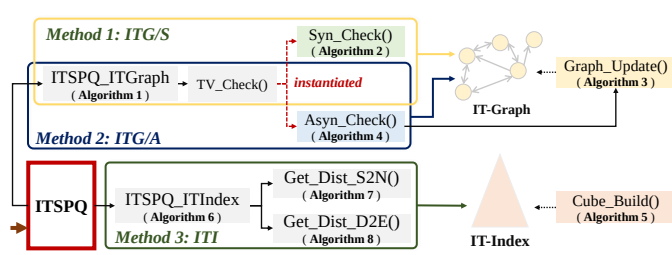


Fig. 5: Different Methods for ITSPQ Processing

The overall framework of ITI is given in Algorithm 6. 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 $path^*$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2021.3076144, IEEE Transactions on Knowledge and Data Engineering

Algorithm 5 `Cube_Build`(N_i)

Input: A node N_i
Output: The distance cube for node N_i
1: initialize cube $\mathsf{DC} : (door \times door \times [t_s, t_e]) \rightarrow (dist, \phi)$
2: **for** each access door $d_i \in N_i$ **do**
3: **for** each door $d_k \in N_i$ **do**
4: `Cell_Build`(d_i, d_k, DC); `Cell_Build`(d_k, d_i, DC)
5: **function** `Cell_Build`(d_s, d_t, DC)
6: $t_1 \leftarrow 0$; $t_2 \leftarrow 0$
7: **while** $t_2 < 24{:}00$ **do**
8: initialize a min-heap H; initialize set $R \leftarrow \varnothing$
9: **for** each door $d_i \in \pi_D(G_{IT}.E)$ **do**
10: **if** $d_i \neq d_s$ **then**
11: $d_i.dist \leftarrow \infty$; $d_i.t_{arr} \leftarrow \infty$, $d_i.t_l \leftarrow \infty$
12: **else**
13: $d_i.dist \leftarrow 0$; $d_i.t_{arr} \leftarrow t_1$; $d_i.t_l \leftarrow t_1$
14: $enheap(H, \langle d_i, d_i.dist \rangle)$
15: **while** H is not empty **do**
16: $\langle d_i, d_i.dist \rangle \leftarrow$ deheap(H)
17: **if** $d_i.dist = \infty$ **then**
18: $t_2 \leftarrow min(R)$; $\mathsf{DC}[d_s, d_t, [t_1, t_2)] \leftarrow (null, \infty)$
19: **break**
20: **if** $d_i = d_t$ **then**
21: $\phi \leftarrow$ concatenate shortest paths from d_s to d_i
22: $t_2 \leftarrow d_i.t_l$
23: $\mathsf{DC}[d_s, d_t, [t_1, t_2)] \leftarrow (d_i.dist, \phi)$
24: **break**
25: mark d_i as visited
26: **if** $d_i.t_{arr} \notin d_i.ATIs$ **then**
27: $t_o \leftarrow$ `Get_Next_Open_Time`($d_i, d_i.t_{arr}$)
28: $R.add(t_o - d_i.dist/velocity)$
29: **continue**
30: **for** each partition $v \in D2P_{\sqsupset}(d_i)$ **do**
31: **if** $d_i \neq d_t$ and $v.d\text{-}type$ is PRD **then continue**
32: **for** each unvisited door $d_j \in P2D_{\sqsubset}(v)$ **do**
33: **if** $d_i.dist + DM(v, d_i, d_j) < d_j.dist$ **then**
34: $d_j.dist \leftarrow d_i.dist + DM(v, d_i, d_j)$
35: $d_j.t_{arr} \leftarrow DM(v, d_i, d_j)/velocity + d_i.t_{arr}$
36: **if** $d_j.t_{arr} \notin d_j.ATIs$ **then** $d_j.t_l \leftarrow d_i.t_l$
37: **else**
38: $t_c \leftarrow$ `Get_Next_Close_Time`($d_j, d_j.t_{arr}$)
39: $d_j.t_l \leftarrow min(t_c - d_j.dist/velocity, d_i.t_l)$
40: $enheap(H, \langle d_j, d_j.dist \rangle)$
41: $t_1 \leftarrow t_2$

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.

`Get_Dist_S2N` (Algorithm 7) 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, $\mathtt{TSD}(s, d, t_c)$ 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 6 `ITSPQ_Index`(p_s, p_t, t, $Index_{IT}$)

Input: Start point p_s, target point p_t, query time t, and $Index_{IT}$
Output: The valid shortest path from p_s to p_t at t
1: $N_{LCA} \leftarrow LCA(\mathtt{Leaf}(p_s), \mathtt{Leaf}(p_t))$ in $Index_{IT}$
2: $N_s \leftarrow$ children of $N_{LCA} \cap$ ancestors of $\mathtt{Leaf}(p_s)$
3: $N_t \leftarrow$ children of $N_{LCA} \cap$ ancestors of $\mathtt{Leaf}(p_t)$
4: $dist^* \leftarrow \infty$; $path^* \leftarrow null$
5: $S2N \leftarrow$ `Get_Dist_S2N`(p_s, N_s, t)
6: **for** $d_i \in AD_{\sqsubset}(N_s)$ **do**
7: $\Delta t_1 \leftarrow S2N[d_i]/velocity + t$
8: **if** $\Delta t_1 \notin d_i.ATIs$ or $d_i.d\text{-}type$ is PRD **then continue**
9: **for** $d_j \in AD_{\sqsupset}(N_t)$ **do**
10: $\Delta t_2 \leftarrow N_{LCA}.\mathsf{DC}[d_i, d_j, \Delta t_1]/velocity + \Delta t_1$
11: **if** $\Delta t_2 \notin d_j.ATIs$ or $d_j.d\text{-}type$ is PRD **then continue**
12: $d2e \leftarrow$ `Get_Dist_D2E`(d_j, p_t, $t + \Delta t_2$)
13: $dist \leftarrow S2N[d_i] + N_{LCA}.\mathsf{DC}[d_i, d_j, \Delta t_1] + d2e$
14: **if** $dist < dist^*$ **then**
15: $dist^* \leftarrow dist$
16: $path^* \leftarrow$ concatenate shortest paths of
17: $p_s \rightarrow d_i$, $d_i \rightarrow d_j$ and $d_j \rightarrow p_t$
18: **return** $path^*$

Algorithm 7 `Get_Dist_S2N`(s, N_s, t_c)

Input: A point s, a node N_s, and current time t_c
Output: The shortest distance from s to N_s at t_c
1: initialize an array $S2N : door \rightarrow dist$
2: $N_c \leftarrow \mathtt{Leaf}(s)$
3: $PN_c \leftarrow$ the parent node of $\mathtt{Leaf}(s)$
4: **while** $N_c \neq N_s$ **do**
5: $CAD_{\sqsubset}(N_c) \leftarrow AD_{\sqsubset}(N_c)$
6: **for** d_i in $AD_{\sqsubset}(N_c)$ **do**
7: **if** $\mathtt{TSD}(s, d_i, t_c)/velocity + t_c \notin d_i.ATIs$ **then** $CAD_{\sqsubset}(N_c) \setminus d_i$
8: **if** $d_i.d\text{-}type$ is PRD and $d_i \notin P2D(\mathsf{P}(s))$ **then** $CAD_{\sqsubset}(N_c) \setminus d_i$
9: **for** each unmarked $d \in AD_{\sqsubset}(PN_c)$ **do**
10: mark d; compute $\mathtt{TSD}(s, d, t_c)$
11: **if** $d \in AD_{\sqsubset}(N_s)$ **then** $S2N[d] \leftarrow \mathtt{TSD}(s, d, t_c)$
12: $N_c \leftarrow PN_c$; $PN_c \leftarrow$ the parent node of PN_c
13: **return** $S2N$
14: **function** `TSD`(pt, d, t_c)
15: **return** the cached result if computed
16: $N_d \leftarrow d$'s current corresponding node
17: **if** pt is a door **then return** $N_d.\mathsf{DC}[pt, d, t_c]$
18: **if** N_d is a leaf node **then**
19: **if** $d \in \mathsf{P}(pt)$ **then return** $|pt, d|_E$
20: **else**
21: $D_o \leftarrow$ obtain doors that one can leave $\mathsf{P}(pt)$ from pt at t_c
22: **if** $D_o \neq \varnothing$ **then**
23: **return** $\min_{d_j \in D_o}(|pt, d_j|_E + \mathtt{TSD}(d_j, d, t_c))$
24: **else**
25: **return** ∞
26: **else**
27: $CN_d \leftarrow N_d$'s child node that contains pt
28: **return** $\min_{d_j \in CAD_{\sqsubset}(CN_d)} \left(\mathtt{TSD}(pt, d_j, t_c) + \mathtt{TSD}(d_j, d, t_c)\right)$

Function `TSD` (lines 14–28) computes the shortest distance as

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2021.3076144, IEEE Transactions on Knowledge and Data Engineering

follows. If the distance 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.

Get_Dist_D2E (Algorithm 8) 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 7. 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).

Algorithm 8 $\texttt{Get\_Dist\_D2E}(d, e, t_c)$

Input: A point s, a point e, and current time t_c
Output: 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** $d_i \in AD_{\sqcupset}(N_c)$ **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 $d_j \in AD_{\sqcupset}(CN_c)$ **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** $\min_{d_j \in CAD_{\sqcupset}(CN_d)}\left(\texttt{TSD2}(d, d_j, t_c) + \texttt{TSD2}(d_j, pt, t_c)\right)$

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 7 but the direction is reversed such that the search starts from a door to a point in the leaf node.

Note that Algorithms 7 and 8 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 6. *Assuming the same* ITSPQ *from p_1 to p_2 as in Example 4, 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.*

4.3 Complexity Analysis

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

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

5 EXPERIMENTAL STUDIES

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

5.1 Results on Synthetic Data

5.1.1 Settings

Indoor Space. Based on a real-world floorplan [2], we generate a multi-floor indoor space where each floor takes 1368m × 1368m. The irregular hallways are decomposed into smaller but regular partitions[2]. As a result, we obtain 141 partitions and 224 (virtual)

2. The decomposition algorithm is given in [29].

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2021.3076144, IEEE Transactions on Knowledge and Data Engineering

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 20*m* 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.

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 5.1.3. Table 3 lists the parameter settings in our experiments, where the default values are shown in bold.

TABLE 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], [14], [22], [23] 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.

5.1.2 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.

Effect of Number of Floors. We vary the number of floors and report the corresponding construction time and index size in Figures 6 and 7, 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.

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 Figures 8 and 9, 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 Figure 10, 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 Figure 11 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 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%.

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.

5.1.3 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.

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.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2021.3076144, IEEE Transactions on Knowledge and Data Engineering

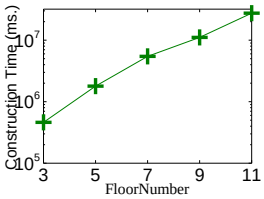


Fig. 6: Time vs. Floor

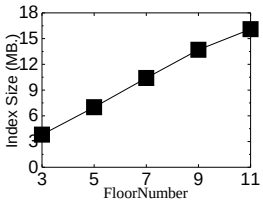


Fig. 7: Size vs. Floor

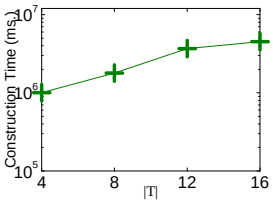


Fig. 8: Time vs. T

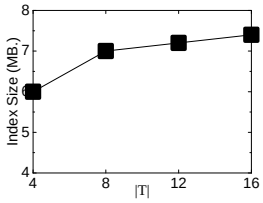


Fig. 9: Size vs. T

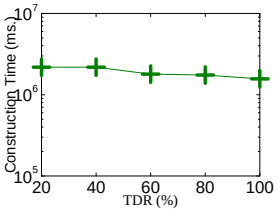


Fig. 10: Time vs. TDR

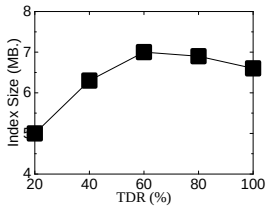


Fig. 11: Size vs. TDR

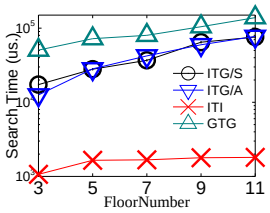


Fig. 12: Time vs. Floor

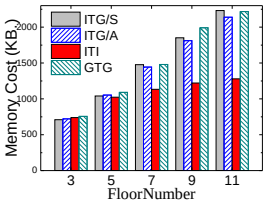


Fig. 13: Memory vs. Floor

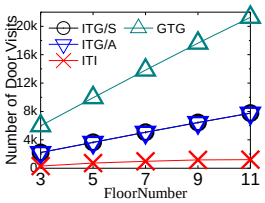


Fig. 14: NDV vs. Floor

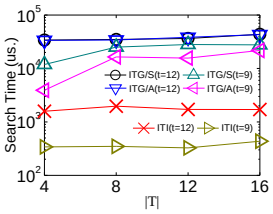


Fig. 15: Time vs. T

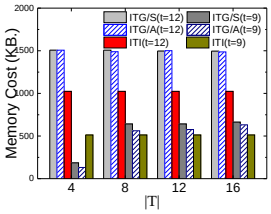


Fig. 16: Memory vs. T

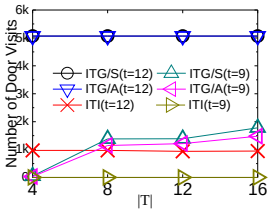


Fig. 17: NDV vs. T

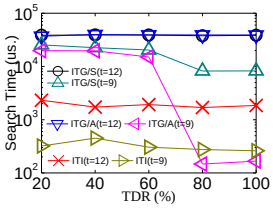


Fig. 18: Time vs. TDR

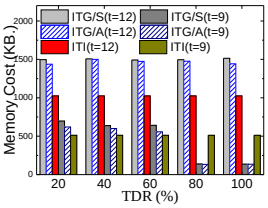


Fig. 19: Memory vs. TDR

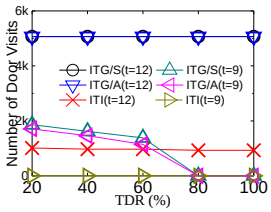


Fig. 20: NDV vs. TDR

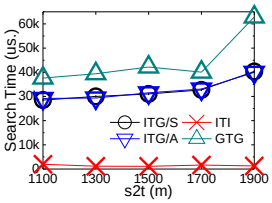


Fig. 21: Time vs. s2t

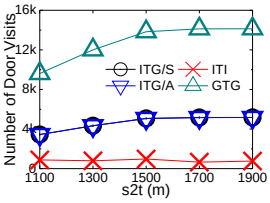


Fig. 22: NDV vs. s2t

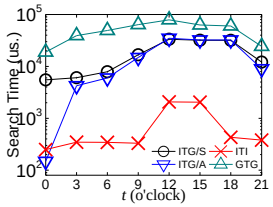


Fig. 23: Time vs. t

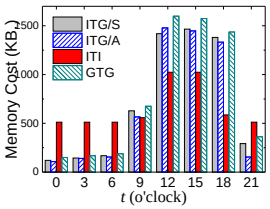


Fig. 24: Memory vs. t

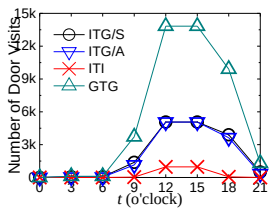


Fig. 25: NDV vs. t

Effect of Number of Floors. Referring to Figure 12, 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 Figure 14. Referring to the memory consumption in Figure 13, 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).

Effect of $|T|$. Referring to Figure 15, 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 Figure 16, 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 Figure 17 exhibit trends consistent with those in the search time reported in Figure 15. Our experiments show that GTG always performs the worst when varying $|T|$. We exclude GTG in Figures 15, 16 and 17 to avoid distraction.

Effect of TDR. Referring to Figures 18 and 20, 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 Figure 19, ITI's memory cost is less than ITG/S and ITG/A. However, when TDR = 80% or 100%

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2021.3076144, IEEE Transactions on Knowledge and Data Engineering

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.

Effect of s2t. When we increase s2t, each method's search time increases slightly, as shown in Figure 21. A similar trend is seen for the NVDs reported in Figure 22. 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 Figure 23, 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 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 Figure 24, 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 Figure 25, 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.

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.

5.1.4 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 Figures 26 and 27. 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.

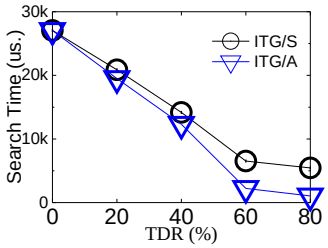


Fig. 26: Time vs. TDR

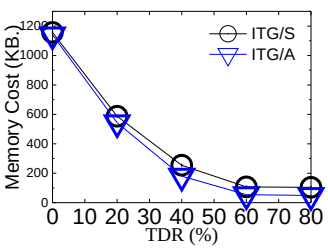


Fig. 27: Memory vs. TDR

5.2 Results on Real Data

We collect a dataset with real indoor topology and temporal variation information from a seven-floor, 2700m × 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 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. Figure 28 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 Figure 29, the results are consistent with the counterparts reported in Figure 11. 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 Figure 30. 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 Figure 31, 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 Figure 32, 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.

6 RELATED WORK

Indoor Spatial Queries. Indoor spaces feature multiple entities like doors, walls, and rooms, altogether forming a complex topol-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2021.3076144, IEEE Transactions on Knowledge and Data Engineering

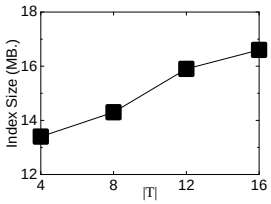


Fig. 28: Index Size vs. T

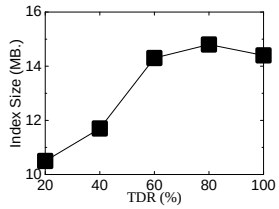


Fig. 29: Index Size vs. TDR

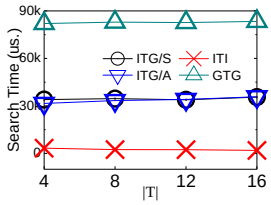


Fig. 30: Time vs. T

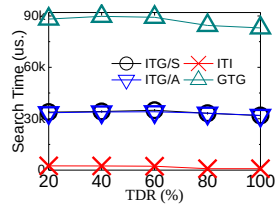


Fig. 31: Time vs. TDR

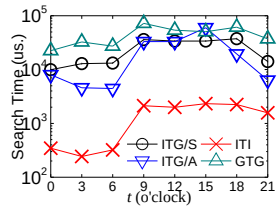


Fig. 32: Time vs. t

ogy that complicates distance-aware queries. Becker et al. [7] 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. [15] 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. [31] 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 [11] define a weighted indoor routing graph with semantic information to create a detailed and user-adaptive model for route search. Lu et al. [19] 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. [26] design IP-tree and VIP-tree that enable more aggressive pruning. VIP-tree also supports indoor trip planning based on neighbour expansion [25]. Luo et al. [20] 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. [4] 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 [28], [29], [30], [32] 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 [21]. Huo et al. [14] analyze and evaluate shortest-path queries on evolving social graphs. Semertzidis et al. [22] study the historical reachability queries on evolving graphs. In the same setting, Semertzidis et al. [23] study three general types of historical queries, namely, historical graph queries, historical time queries and historical top-k queries. Semertzidis et al. [24] use a compressed time neighbourhood and path index to find the durable matches of an input pattern on the temporal graphs. Akiba et al. [3] study the shortest-path distance queries on large time-evolving graphs by using two dynamic indexing schemes. Huang et al. [13] investigate the properties of temporal DFS and BFS, and propose efficient algorithms for route query in a temporal graph. Hirsch et al. [12] 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. [9] 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. [5] propose an adaptive approach to solve the dynamic shortest path problem. In the same setting, Ardakani et al. [6] design an A$*$ algorithm using the decremental approach to speed up the shortest path query processing in dynamic networks. Wei et al. [27] 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 [3], [12], [13], [14], [22], [23], [24] 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 [5], [6], [9], [27] 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.

7 Conclusion and Future Work

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.

REFERENCES

[1] Human average walking speed. https://en.wikipedia.org/wiki/Walking. Accessed October 1, 2019.

[2] Floorplan for a shopping mall. https://www.deviantart.com/mjponso/art/Floor-Plan-for-a-Shopping-Mall-86396406. Accessed October 1, 2019.

[3] T. Akiba, Y. Iwata, and Y. Yoshida. Dynamic and historical shortest-path distance queries on large evolving networks by pruned landmark labeling. In *WWW*, pages 237–248, 2014.

[4] 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*, pages 2841–2857, 2020.

[5] M K. Ardakani, and L. Sun. Decremental algorithm for adaptive routing incorporating traveler information. *Computers & operations research*, pages 3012–3020, 2012.

[6] 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*, pages 299–30, 2015.

[7] C. Becker and F. Dürr. On location models for ubiquitous computing. *Personal and Ubiquitous Computing*, 9(1):20–31, 2005.

[8] M A. Cheema. Indoor location-based services: challenges and opportunities. *SIGSPATIAL Special*, 10(2):10–17, 2018.

[9] B. Ding, J X. Yu, and L. Qin. Finding time-dependent shortest paths over large graphs. In *EDBT*, pages 205–216, 2008.

[10] Z. Feng, T. Liu, H. Li, H. Lu, L. Shou and J. Xu. Indoor Top-*k* Keyword-aware Routing Query. In *ICDE*, pages 1213–1224, 2020.

[11] M. Goetz and A. Zipf. Formal definition of a user-adaptive and length-optimal routing graph for complex indoor environments. *Geo-Spatial Information Science*, 14(2):119–128, 2011.

[12] M J. Hirsch, A. Sadeghnejad and H. Ortiz-Peña. Shortest paths for routing information over temporally dynamic communication networks. In *MILCOM*, pages 587–591, 2017.

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

[14] W. Huo and V J. Tsotras. Efficient temporal shortest path queries on evolving social graphs. In *SSDBM*, pages 1–4, 2014.

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

[16] H. Li, H. Lu, L. Shou, G. Chen, and K. Chen. Finding most popular indoor semantic locations using uncertain mobility data. *IEEE Trans. Knowl. Data Eng.*, 31(11): 2108–2123, 2018.

[17] H. Li, H. Lu, L. Shou, G. Chen, and K. Chen. In search of indoor dense regions: An approach using indoor positioning data. *IEEE Trans. Knowl. Data Eng.*, 30(8): 1481–1495, 2018.

[18] 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 *ICDE*, pages 2014–2017, 2020.

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

[20] W. Luo, P. Jin, and L. Yue. Time-constrained sequenced route query in indoor spaces. In *APweb*, pages 129–140, 2016.

[21] O. Michail. An introduction to temporal graphs: An algorithmic perspective. *Internet Mathematics*, 12(4):239–280, 2016.

[22] K. Semertzidis, E. Pitoura, and K. Lillis. TimeReach: Historical reachability queries on evolving graphs. In *EDBT*, pages 121–132, 2015.

[23] K. Semertzidis and E. Pitoura. Time traveling in graphs using a graph database. In *EDBT/ICDT workshops*, 2016.

[24] K. Semertzidis and E. Pitoura. Top-*k* durable graph pattern queries on temporal graphs. *IEEE Trans. Knowl. Data Eng.*, 31(1):181–194, 2019.

[25] Z. Shao, M A. Cheema, and D. Taniar. Trip planning queries in indoor venues. *The Computer Journal*, 61(3):409–426, 2017.

[26] Z. Shao, M A. Cheema, D. Taniar, and H. Lu. VIP-tree: an effective index for indoor spatial queries. *PVLDB*, 10(4):325–336, 2016.

[27] V J. Wei, R C. Wong, and C. Long. Architecture-Intact Oracle for Fastest Path and Time Queries on Dynamic Spatial Networks. In *SIGMOD*, pages 1841–1856, 2020.

[28] X. Xie, H. Lu, and T B. Pedersen. Distance-aware join for indoor moving objects. *IEEE Transactions on Knowledge and Data Engineering*, 27(2):428–442, Feb 2015.

[29] X. Xie, H. Lu, and T B. Pedersen. Efficient distance-aware query evaluation on indoor moving objects. In *ICDE*, pages 434–445, 2013.

[30] B. Yang, H. Lu, and C S. Jensen. Probabilistic threshold k nearest neighbor queries over moving objects in symbolic indoor space. In *EDBT*, pages 335–346, 2010.

[31] W. Yuan and M. Schneider. iNav: An indoor navigation model supporting length-dependent optimal routing. In *Geospatial thinking*, pages 299–313, 2010.

[32] W. Yuan and M. Schneider. Supporting continuous range queries in indoor space. In *MDM*, pages 209–214, 2010.

Tiantian Liu is a PhD student in the Department of Computer Science, Aalborg University, Denmark. She received the BSc and MSc degrees from Jilin University, China. Her research interests include spatio-temporal data analytics and data management.

Zijin Feng received the BSc degree in computer science from Hong Kong Baptist University, in 2019. He is currently a PhD student at Department of Systems Engineering and Engineering Management, the Chinese University of Hong Kong. His research interests include graph query processing, graph data management and location-based services.

Huan Li is a postdoctoral research fellow in the Department of Computer Science, Aalborg University, Denmark. He received the PhD degree in computer science from Zhejiang University, China. His research interests include spatio-temporal data analytics and mobile/pervasive computing. He is a member of the ACM.

Hua Lu is a Professor in the Department of People and Technology, Roskilde University, Denmark. He received the BSc and MSc degrees from Peking University, China, and the PhD degree in computer science from National University of Singapore. His research interests include database and data management, geographic information systems, and mobile computing. He has served as PC cochair or vice chair for ISA 2011, MUE 2011 and MDM 2012, demo chair for SSDBM 2014, and PhD forum cochair for MDM 2016. He has served on the program committees for conferences such as VLDB, ICDE, CIKM, DASFAA, ACM SIGSPATIAL, SSTD, MDM, PAKDD, and APWeb. He is a senior member of the IEEE.

Muhammad Aamir Cheema received the PhD degree from UNSW Australia, in 2011. He is a senior lecturer in the Clayton School of Information Technology, Monash University, Australia. He is the recipient of the 2012 Malcolm Chaikin Prize for Research Excellence in engineering, 2013 Discovery Early Career Researcher Award, and the 2014 Dean's Award for Excellence in Research by an Early Career Researcher. His PhD thesis was nominated for the SIGMOD Jim Gray Doctoral Dissertation Award and ACM Doctoral Dissertation Competition. He has won two CiSRA best research paper awards (in 2009 and 2010), two invited papers in the special issue of IEEE TKDE on the best papers of ICDE (2010 and 2012), and two best paper awards at WISE 2013 and ADC 2010, respectively. He served as PC co-chair for ADC 2015, ADC 2016, 8th ACM SIGSPATIAL Workshop ISA 2016, and WWW International Workshop on Social Computing 2017, proceedings chair for DASFAA 2015, tutorial co-chair for APWeb 2017, and publicity co-chair for ACM SIGSPATIAL 2017.

Hong Cheng received the PhD degree from the University of Illinois at Urbana-Champaign, in 2008. She is an associate professor in the Department of Systems Engineering and Engineering Management, Chinese University of Hong Kong. Her research interests include data mining, database systems, and machine learning. She received research paper awards at ICDE'07, SIGKDD'06, and SIGKDD'05, and the certificate of recognition for the 2009 SIGKDD Doctoral Dissertation Award. She received the 2010 Vice-Chancellor's Exemplary Teaching Award at the Chinese University of Hong Kong.

Jianliang Xu received the BEng degree in computer science and engineering from Zhejiang University, Hangzhou, China, and the PhD degree in computer science from the Hong Kong University of Science and Technology. He is a professor in the Department of Computer Science, Hong Kong Baptist University. He held visiting positions with Pennsylvania State University and Fudan University. His research interests include big data management, mobile computing, and data security and privacy. He has published more than 150 technical papers in these areas. He has served as a program cochair/vice chair for a number of major international conferences including IEEE ICDCS 2012, IEEE CPSNA 2015, and APWeb-WAIM 2018. He is an associate editor of the IEEE Transactions on Knowledge and Data Engineering and the Proceedings of the VLDB Endowment 2018.