Shortest Path Queries for Indoor Venues with Temporal Variations

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

Published in:
The 36th IEEE International Conference on Data Engineering (ICDE 2020)

Creative Commons License
CC BY 4.0

Publication date:
2020

Document Version
Version created as part of publication process; publisher's layout; not normally made publicly available

Link to publication from Aalborg University

Citation for published version (APA):
Shortest Path Queries for Indoor Venues with Temporal Variations

Tiantian Liu† Zijin Feng‡ Huan Li‡ Hua Lu‡ Muhammad Aamir Cheema§ Hong Cheng‡ Jianliang Xu‡

†Department of Computer Science, Aalborg University, Denmark
‡Department of Systems Engineering and Engineering Management, The Chinese University of Hong Kong, Hong Kong
§Faculty of Information Technology, Monash University, Australia

Abstract—Indoor shortest path query (ISPQ) is of fundamental importance for indoor location-based services (LBS). However, existing ISPQs 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, respectively. We experimentally evaluate the proposed techniques using synthetic data. The results show that our methods are efficient.

I. INTRODUCTION

With 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, tracking staff and equipment in hospitals, and location-based shopping assistance for customers [3], [5], [7], [8], [11], [13].

Shortest path/distance queries [2], [9], [12], [14] are fundamental in many indoor location-based services. However, most existing techniques assume that the indoor topology does not change with time. As a matter of fact, doors may be restricted at certain times of the day, e.g., doors leading to patient wards in a hospital may only open during visiting hours. Such temporal variations clearly change the indoor topology, which entails indoor navigation aware of topological changes. Some previous works have studied temporal graphs [4], [6], [10], but those techniques do not consider complex topology and semantic information in indoor space.

In this paper, we propose to study indoor temporal-variation aware shortest path query (ITSPQ) which returns a shortest path from a source point to a target point such that navigation through private partitions is not allowed and 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 pre-computed and materialized door-to-door distances become invalid when one or more doors open or close at certain times.

To address these challenges, we propose an indoor temporal-variation graph (IT-GRAph) which captures the indoor topology, geometric information, and temporal variation information in a composite structure. The indoor temporal variations are represented by time intervals. Figure 1 shows an example indoor space where the doors may be open and closed at different times, as listed in Table I. In our setting, we use (open-time, close-time) to denote an active time interval (ATI) of a door. Thus, [8:00, 16:00) means a door is opened at 8:00 and closed at 16:00. If a door features multiple ATIs, we use an array to store them.

Fig. 1: An Example of Indoor Floor Plan

We formulate our research problem as follows.

Research Problem (Indoor Temporal-Variation Aware Shortest Path Query (ITSPQ)). Given a start point $p_s$, a target point $p_t$, and a current timestamp $t$, an indoor temporal-variation aware shortest path query $\text{ITSPQ}(p_s, p_t, t)$ 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 $t + \Delta_t^2$, where $\Delta_t$ is the walking time from $p_s$ to $d_i$ and it is computed based on human’s average walking speed [1] — $5 \text{km/h}$;
2) The path should not go through any private partition except the private partitions that contain $p_s$ and/or $p_t$.

Example 1. Given a query $\text{ITSPQ}(p_3, p_5, 9:00)$, we consider two candidate indoor paths, i.e., $(p_3, d_{15}, d_{16}, p_5)$ with

---

1Private partitions are not public for all users, e.g., private offices in an office building, security zones in airports, and storage areas in a mall.

2In 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.
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 a result. In contrast, another query ITSPQ\((p_3, p_4, 23:30)\) returns null because \(d_{18}\) is close at that time and no path can meet both rules in the problem definition.

II. ITSPQ PROCESSING

A. Indoor Temporal-Variation Graph

To integrate the temporal variations of doors into the indoor topology, we design an indoor temporal-variation graph (IT-G) \(G_{IT}(V, E, L_v, L_e)\) where

1) \(V\) is the set of vertices such that each vertex \(v \in V\) is an indoor partition.

2) \(E\) is the set of directed edges such that 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 \(\pi_0(E)\) to denote the set of doors associated with the edges of \(E\).

3) \(L_v\) is the set of vertex labels, each being a 3-tuple \((d_{let}, p_{type}, DM)\) where \(d_{let}\) identifies the partition in the vertex, \(p_{type}\) = \{PB, PRP\} indicates if the partition is a public partition (PB) or a private partition (PRP), and \(DM\) is a distance matrix [9] 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 being a 3-tuple \((d_{let}, d_{type}, ATIs)\) where \(d_{let}\) identifies the door on the edge, \(d_{type}\) = \{PR, PRD\} indicates if the door is a public (PRD) or private (PRD) door, and \(ATIs\) is the door’s ATIs.

The IT-G corresponding to Figure 1 is depicted in Figure 2. We use a door table and a partition table to store \(L_v\) and \(L_e\) in IT-G, respectively. Referring to the tables in Figure 2, a record \((d_1, PRD, [(6:00, 23:30)])\) means \(d_1\) is a private door open from 6:00 to 23:30, and \(v_{16}\) is a public partition and the distance between its doors \(d_3\) and \(d_{17}\) is 2m.

Following the previous work [9], \(P2D(v_k)\) 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, \(P2D(d_i)\) gives the set of enterable doors through which one can enter partition \(v_k\), \(P2D(d_i)\) gives the set of leaveable doors through which one can leave partition \(v_k\), \(D2P(d_i)\) gives the set of partitions that one can enter through door \(d_i\), and \(D2P(d_i)\) gives those that one can leave through door \(d_j\). These mappings can be easily obtained based on the connectivity information in IT-G. Referring to Figure 2, we have \(D2P(d_1) = \{v_1,v_{16}\}\), \(D2P(d_3) = v_3\), and \(D2P(d_5) = v_{16}\). Also, we have \(P2D(v_3) = P2D(v_{16}) = \{v_1,d_1,d_2,d_3,d_5,d_6\}\) whereas \(P2D(v_{16}) = \{d_1,d_2,d_3,d_5,d_6\}\).

<table>
<thead>
<tr>
<th>Door. ATIs</th>
<th>Door. ATIs</th>
</tr>
</thead>
<tbody>
<tr>
<td>(d_1, (5:00, 23:00))</td>
<td>(d_2, (8:00, 16:00))</td>
</tr>
<tr>
<td>(d_3, (6:00, 23:00))</td>
<td>(d_4, (9:00, 18:00))</td>
</tr>
<tr>
<td>(d_5, (6:30, 23:00))</td>
<td>(d_6, (8:00, 16:00))</td>
</tr>
<tr>
<td>(d_7, (6:00, 23:30))</td>
<td>(d_8, (9:00, 18:00))</td>
</tr>
<tr>
<td>(d_9, [(0:00, 6:00), (6:30, 23:00)])</td>
<td>(d_{10}, (8:00, 16:00))</td>
</tr>
<tr>
<td>(d_{11}, (5:00, 23:00))</td>
<td>(d_{12}, (5:00, 23:00))</td>
</tr>
<tr>
<td>(d_{13}, (5:00, 17:00), (18:00, 23:00))</td>
<td>(d_{14}, (0:00, 24:00))</td>
</tr>
<tr>
<td>(d_{15}, (8:00, 16:00))</td>
<td>(d_{16}, (8:00, 17:00))</td>
</tr>
<tr>
<td>(d_{17}, (0:00, 24:00))</td>
<td>(d_{18}, (10:00, 25:00))</td>
</tr>
<tr>
<td>(d_{19}, (8:00, 16:00))</td>
<td>(d_{20}, (5:00, 23:00))</td>
</tr>
</tbody>
</table>

Table I: Active Time Intervals (ATIs) of Doors

B. Algorithms for ITSPQ Processing

The overall framework for processing ITSPQ based on IT-G is presented in Algorithm 1. It 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 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 \(\infty\), meaning all remaining unvisited doors cannot get to \(p_t\), “no such routes” is returned (line 10). If \(d_i\) is equal to \(p_t\), 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 \(d_i\). Particularly, if \(d_i\) equals \(p_s\), \(v\) is \(p_s\)’s covering partition \(P(p_s)\). 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 \(p_s\)’s covering partition \(P(p_s)\) (line 20), it means that 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 leaveable 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 a private partition (line 28). Then, the current path distance \(dist[d_{j_{cur}}]\) from \(p_s\) to \(d_j\) is obtained as the sum of \(dist[d_{j_{cur}}]\) and distance from \(d_i\) to \(d_{j_{cur}}\) through \(v\). Next, the framework calls a function \(TV\_Check(d_{j_{cur}}, dist[d_{j_{cur}}], t)\) to validate if \(d_j\) is open at the arrival time relative to the query time \(t\) (line 30). Two different strategies, namely \(Syn\_Check()\) (Algorithm 2) and
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 ITSPQ_ITGraph\(p_s, p_t, t, G_{IT}\)**

1: initialize a min-heap \(H\)
2: for each door \(d_i \in \pi_G(G_{IT}, E)\) do
3: \(dist[d_i] \leftarrow \infty\)
4: enheap\(H, (d_i, dist[d_i])\)
5: \(prev[d_i] \leftarrow \text{null} \)
6: \(dist[p_i] \leftarrow 0\); enheap\(H, (p_i, dist[p_i])\)
7: \(dist[p_i] \leftarrow \infty\); enheap\(H, (p_i, dist[p_i])\)
8: while \(H\) is not empty do
9: \(d, dist[d]\) \(\leftarrow\) deheap\(H\)
10: if \(dist[d_i] = \infty\) then return no such routes
11: if \(d_i = p_t\) then
12: \(path \leftarrow p_t\)
13: while \(prev[d_i] \neq p_i\) do
14: \(path \leftarrow prev[d_i] + "\), + path\)
15: \(d_i \leftarrow prev[d_i]\)
16: \(path \leftarrow p_i + "\), + path\)
17: return \(path\)
18: if \(d_i = p_t\) then \(v \leftarrow P(p_t)\) else \(v \leftarrow DP2\_G(d_i)\) \(\setminus\) visited partitions
19: mark \(d_i\) and \(v\) as visited
20: if \(d_i \in DP2\_G(P(p_i))\) then
21: if \(dist[d_i] + |d_i, p_i|_E < dist[p_i]\) then
22: \(dist[p_i] \leftarrow dist[d_i] + |d_i, p_i|_E\)
23: enheap\(H, (p_i, dist[p_i])\)
24: \(prev[p_i] \leftarrow (v, d_i)\)
25: else for each unvisited door \(d_j \in DP2\_G(v)\) do
26: \(v' \leftarrow DP2\_G(d_j)\) \(\setminus v\)
27: if \(v'\)’s type is PRP then continue
28: \(dist \leftarrow dist[d_i] + DM(v_i, d_i)\)
29: if TV\_Check\(d_i, dist, t\) then continue
30: if \(dist < dist[d_i]\) then
31: \(dist[d_i] \leftarrow dist\)
32: enheap\(H, (d_i, dist[d_i])\)
33: \(prev[d_i] \leftarrow (v, d_i)\)

**Synchronous Check.** The idea 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 query time \(t\) plus the travel time \((dist/velocity)\) to go through the distance \(dist\) from \(p_o\) to \(d\) (line 1). The function returns false if \(t_{arr}\) is not in the ATIs, and true otherwise.

**Algorithm 2 Syn_Check\(d, dist, t\)**

1: \(t_{arr} \leftarrow t + dist/velocity\)
2: if \(t_{arr} \notin d\)’s ATIs then return false else return true

**Asynchronous Check.** Synchronous check needs to validate each encountered door by comparing the arrival time with the door’s ATIs. However, in real-world scenarios, the temporal variation of doors in IT-GRAH can only happen at several particular open or close times. We call such time points as checkpoints. Moreover, the topology information will not change between two consecutive checkpoints. An alternative checking strategy is to directly refer to a time-dependent IT-GRAH that only keeps all currently open doors. The information of IT-GRAH only needs to be updated asynchronously at the next checkpoint. Given the set \(T\) of checkpoints, the graph updating at a current time \(t\) is presented in Algorithm 3. First, it initializes a new graph \(G_{IT}'\) using the initial graph \(G_{IT}\) that keeps the original indoor topology without considering temporal variations. Next, it searches the previous checkpoint \(cp\) relative to \(t\) (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 modifies the mapping information for that door and its corresponding partitions (lines 4–7). Finally, it returns \(cp\) and the new graph \(G_{IT}'\).

**Algorithm 3 Graph_Update\(t, T\)**

1: \(G_{IT}' \leftarrow G_{IT}\)
2: \(cp \leftarrow \text{Find_Previous_Checkpoint}(t, T)\)
3: \(D_c \leftarrow \text{Get_Closed_Door}(cp)\)
4: for each door \(d_i \in D_c\) do
5: \(P_{p_i} \leftarrow DP2(p_i)\)
6: for each partition \(v \in P_{p_i}\) do
7: \(P2D2(v) \leftarrow P2D(v) \setminus d_i\)
8: replace \(P2D(v)\) in \(G_{IT}'\) by \(P2D2(v)\)
9: 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 9 in Algorithm 3) and the arrival time \(t_{arr}\) (lines 1–2). Next, if \(t_{arr}\) to reach \(d\) is later than the next checkpoint in \(T\), it updates \(G_{IT}'\) using \(G_{IT}\) returned by Algorithm 3 (lines 4–6). A false is returned to keep consistent with the interface of Algorithm 2 (line 7).**

**Algorithm 4 Asyn_Check\(d, dist, t\)**

1: get the current \(G_{IT}'\) and its corresponding \(cp\) for time \(t\)
2: \(t_{arr} \leftarrow t + dist/velocity\)
3: \(G_{IT}' \leftarrow \text{null} \)
4: if \(t_{arr} > \text{Find_Next_Checkpoint}(cp, T)\) then
5: if \(G_{IT}'\) is null then \((cp, G_{IT}') \leftarrow \text{Graph_Update}(t_{arr}, T)\)
6: \((cp, G_{IT}') \leftrightarrow (cp, G_{IT}')\)
7: return false

Compared to the search with synchronous check, the search using asynchronous check involves reduced versions of IT-GRAH in the outward expansion (lines 18–34 in Algorithm 1), thus pruning some closed doors in advance and reducing the cost of checking temporal variations.

We use ITG/S to denote the search method using synchronous check, and ITG/A the one using asynchronous check. Figure 3 illustrates the two methods.

**Fig. 3: Different Methods for ITSPQ Processing**

**III. EXPERIMENTAL STUDIES**

Using synthetic data, we evaluate the search efficiency of our proposed methods ITG/S and ITG/A. All experiments are implemented in Java and run on a PC with a 2.30GHz Intel i5 CPU and 16 GB memory.
1) **Settings: Indoor Space.** Using a real-world floorplan, we generate a multi-floor indoor space where each floor takes 1368m $\times$ 1368m. The irregular hallways are decomposed into smaller, regular partitions. As a result, we obtain 141 partitions and 224 (virtual) doors. Every two adjacent floors are connected by four staircases, each having a stairway of 20m long. In the default setting, we use a 5-floor indoor space with 705 partitions and 1120 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. For each door with temporal variation, we assign it with up to three ATIs, each corresponding to a pair of open time and close time selected from $T$.

**Query Instances.** We use a parameter $\delta_{2t}$ to control the indoor distance from the start point $p_s$ to the target point $p_t$ in a query $ITSPQ(p_s, p_t, t)$ as follows. First, we randomly select a point $p_s$ from the indoor space. Second, we find a door $d$ whose indoor distance to $p_s$ approaches $\delta_{2t}$. Then, we expand from $d$ to find a random point $p_t$ whose indoor distance to $p_s$ approaches $\delta_{2t}$. For each setting of $\delta_{2t}$, we generate five pairs of $p_s$ and $p_t$ to form the query instances. In each query instance, 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. Table II lists the parameter settings in our experiments, where the default values are in bold.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>T</td>
</tr>
<tr>
<td>$\delta_{2t}$ (m)</td>
<td>1100, 1300, 1500, 1700, 1900</td>
</tr>
<tr>
<td>$t$</td>
<td>0:00, 2:00, ..., 12:00, ..., 22:00</td>
</tr>
</tbody>
</table>

**Performance Metrics.** We run each query instance ten times, and measure the average running time and memory cost.

2) **Efficiency of Search Methods:** We investigate the search time and memory cost of our proposed methods, i.e., ITG/S and ITG/A, under different parameter settings.

**Effect of $|T|$.** Referring to Figure 4, 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 8:00. At this time, increasing $|T|$ makes more doors be closed, reducing the cost of graph search. As a result, the search of each method becomes faster.

**Effect of $\delta_{2t}$.** When we increase $\delta_{2t}$, each method’s search time increases slightly, as shown in Figure 5.

**Effect of $t$.** We also test the search methods’ performance at different query times ($t$) in a day. Referring to Figure 6, the search time of each method increases when $t$ comes to 10:00, stays stable when $t$ is between 10:00 and 20:00, and then decreases when $t$ is over 20:00. In our setting, a large number of doors have been closed for the time before 10:00 or after 20:00, and the corresponding IT-GRAPH 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 10:00 to 20:00. Between 10:00 and 20:00, the memory costs of all methods stay constant because nearly all doors are open and the indoor topology is relatively stable. After 20:00, the memory costs of all methods decrease as the graph structure becomes simpler.

**Acknowledgement.** This work was supported by IRFD (No. 8022-00366B), HK-RGC (No. 12200819 and 12201018) and ARC (No. FT180100140 and DP180103411).

**REFERENCES**


