Aalborg Universitet

Indoor Top-k Keyword-aware Routing Query

Feng, Zijin; Liu, Tiantian; LI, HUAN; Lu, Hua; Shou, Lidan; Xu, Jianliang

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

DOI (link to publication from Publisher):
10.1109/ICDE48307.2020.00109

Publication date:
2020

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

Indoor Top-k Keyword-aware Routing Query

Zijin Feng[†] Tiantian Liu[‡] Huan Li[‡] Hua Lu[‡] Lidan Shou[§] Jianliang Xu[†]

[†]Department of Computer Science, Hong Kong Baptist University, Hong Kong
[‡]Department of Computer Science, Aalborg University, Denmark
[§]Department of Computer Science, Zhejiang University, China
15251519@life.hkbu.edu.hk, {liutt, lihuan, luhua}@cs.aau.dk, should@zju.edu.cn, xujl@comp.hkbu.edu.hk

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

I. Introduction

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

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

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

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

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

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

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

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

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

perimental results demonstrate the efficiency of our proposals and disclose the respective suitable settings for ToE and KoE.

We make the following contributions in this paper:

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

In addition, we review the related work in Section VI and conclude the paper in Section VII.

II. Problem Formulation

A. Preliminaries

Table I lists the frequently used notations.

TABLE I: Notations

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

A previous work [13] defines mappings that capture indoor topology. In particular, $D2P_{\sqsupset}(d_i)$ gives the set of partitions that one can enter through door d_i and $D2P_{\sqsubset}(d_j)$ gives those that one can leave through door d_j. As a basic step, indoor routing needs to move from one door to another through their common partition. To this end, we have intra-partition **door-to-door distance** for two doors d_i and d_j as

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

Here, $D2P_{\sqsupset}(d_i) \cap D2P_{\sqsubset}(d_j) \neq \varnothing$ means d_i and d_j are in the same partition that one can enter via d_i and leave via d_j. In this case, we measure the Euclidean distance between d_i and d_j. The case of $d_i = d_j$ is special. This happens when one needs to enter a partition due to its keyword relevance but then leave it from the same door for further routing. In this case, we set $\delta_{d2d}(d_i,d_j)$ to be the double of the longest non-loop distance one can reach inside the partition from the pertinent door. Note that δ_{d2d} simplifies the f_{d2d} function [13] such that no partition is explicitly specified.

Moreover, the previous work [13] uses $v(p_i)$ to denote point p_i's host partition, $P2D_{\sqsupset}(v_k)$ the set of *enterable* doors through which one can enter partition v_k, and $P2D_{\sqsubset}(v_k)$ the set of *leaveable* doors through which one can leave partition v_k. Still within a partition, we define **point-to-door distance** and **door-to-point distance**, respectively, as follows. Given a door d_k, two points p_i and p_j, we have

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

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

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

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

B. Principles and Definition of Routing Query

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

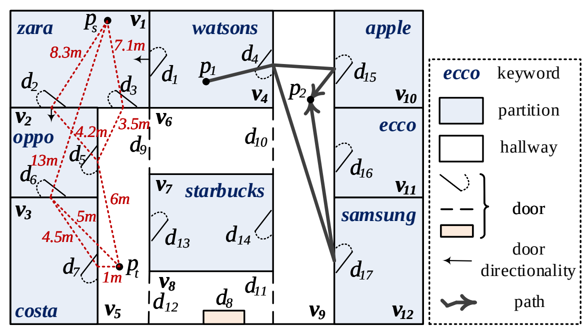

Fig. 1: An Example of Floorplan

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

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

Principle of Regularity. Traditional outdoor routing algorithms [1], [7], [11], [23] usually exclude loops in a route. This regularization avoids endless route searching. However, a regular route in indoor space can have a loop of doors within one-hop. Referring to Fig. 1, anyone who needs to visit partition v_{10} must enter and then leave d_{15}, the only accessible door of v_{10}. Accordingly, the principle of regularity disqualifies a route that contains one or more doors between two identical doors. For example, partial route $(d_{13},d_{14},d_{14},d_{13})$ is not allowed, because of the doors between the two appearances of door d_{13}. This partial route means that one starts from d_{13}, passes v_7 twice and returns to d_{13} again.

***Principle of Diversity*.** The idea of diversifying top-k results [14] inspires us to avoid homogeneous routes in our indoor routing. Back to the example in Fig. 1, suppose a user wants routes from p_s to p_t while covering two keywords *oppo* and *costa*. Several possible routes are listed in Table II. For ease of reading, we insert between each two consecutive route items the partition that connects the two items.

TABLE II: Examples of Routes from p_s to p_t

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

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

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

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

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

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

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

Problem 1 (Indoor Top-k Keyword-aware Routing Query)**.** *Given a start point p_s, a terminal point p_t, a distance constraint Δ, and a query keyword list QW, an* **indoor top-k keyword-aware routing query** *IKRQ(p_s, p_t, Δ, QW, k) returns k regular and prime routes from p_s to p_t in a k-set Θ such that $\forall R \in \Theta$, $\delta(R) \leq \Delta$ and $\Psi(R, \Delta, QW) \geq \Psi(R', \Delta, QW)$ for any route $R' \notin \Theta$ from p_s to p_t with $\delta(R') \leq \Delta$.*

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

III. Ranking Relevant Routes for IKRQ

A. Organization of Indoor Space Keywords

We differentiate two types of keywords associated with indoor partitions. An **identity word** (i-word) identifies the specific name of a partition, while a **thematic word** (t-word) [4] refers to a tag relevant to that partition. A partition can relate to one i-word only but a set of t-words. For a specific indoor venue, i-words can be obtained from floor map or the like, and t-words can be extracted from the semantic descriptions of the indoor partitions or those of the corresponding i-words. For example, i-words in a mall are shop names like *starbucks* and *zara* and function area names like *frontdesk* and *toilet*. Meanwhile, a shop *zara* can be associated with many t-words such as *pants*, *sweater* and *coat*.

Given an indoor venue, we organize its i-words and t-words in two disjoint sets. If a word is in the i-word set W_i, it is excluded from the t-word set W_t to keep the two keyword sets distinct. Given the full set V of partitions in an indoor venue, a **P2I** mapping $P2I(v_k)$ maps a partition $v_k \in V$ to its associated i-word $w_i \in W_i$, and an **I2P** mapping $I2P(w_i)$ maps an i-word $w_i \in W_i$ to a set of relevant partitions. Moreover, an **I2T** mapping $I2T(w_i)$ maps an i-word $w_i \in W_i$ to a set of relevant t-words, and a **T2I** mapping $T2I(w_t)$ maps a t-word $w_t \in W_t$ to a set of relevant i-words.

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

Example 3. *Fig. 2 illustrates parts of the indoor space keyword mappings for the example in Fig. 1. Partition v_3 is mapped to an i-word costa via the P2I mapping. Reversely, we can use the I2P mapping to find that the i-word apple is associated with partition v_{10}. According to the I2T and T2I mappings, t-words laptop and smartphone are relevant to the i-word apple, and the i-word costa is relevant to t-words coffee and mocha. Moreover, v_3's partition words $PW(v_3) = \{costa, \{coffee, mocha, \ldots\}\}$.*

In our organization, i-words act as the pivot between partitions and t-words, as the vocabulary of i-words is much smaller than that of t-words, making the mappings between i-words

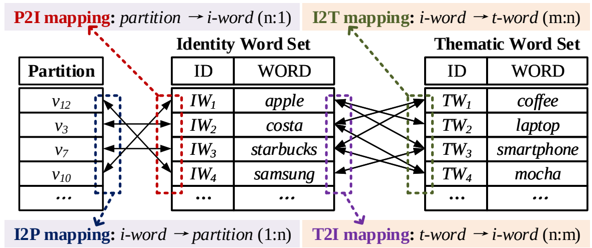

Fig. 2: Indoor Space Keyword Mappings

and partitions more concise. Using the mappings described above, we are able to quantify the keyword relevance between query keywords and routes.

B. Keyword Relevance between Query Keywords and Routes

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

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

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

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

Example 4. *Corresponding to Fig. 2, some partitions and their partition words are listed below.*

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

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

Finally, we can convert the original query keyword list to a list of candidate i-word sets: $\mathbf{K}(QW) = \langle \kappa(latte), \kappa(apple)\rangle = \langle \{(starbucks, 1), (costa, 0.75)\}, \{(apple, 1)\}\rangle$.

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

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

Example 5. *Referring to the route $R = (p_s, d_3, p_t)$ in Fig. 1, for point p_s, we have $PW(v(p_s)).w_i = PW(v_1).w_i = \{zara\}$. Likewise, we have $PW(v(p_t)).w_i = \varnothing$. For door d_3, we have $PW(D2P_{\sqsubset}(d_3)).w_i = PW(v_1).w_i \cup PW(v_5).w_i = \{zara\} \cup \varnothing = \{zara\}$. Consequently, we have $RW(R) = \{zara\}$.*

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

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

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

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

The computation complexity of $\rho(R)$ is $O(mn)$, where m is $|QW|$ and n is the number of i-words in a route R.

Example 6. *Referring to Fig. 1, we have two routes* $R_1 = (p_s, d_2, d_5, d_7, d_7, p_t)$ *and* $R_2 = (d_{15}, d_{15}, d_{14}, d_{13}, d_7, d_6)$ *and a query keyword list* $QW = \{latte, apple\}$. *For route* R_1, *we have* $RW(R_1) = \{zara, oppo, costa\}$, *and* $\kappa(latte).W_i \cap RW(R_1) = \{costa\}$ *with similarity score* 0.75 *and* $\kappa(apple).W_i \cap RW(R_1) = \emptyset$, *thus* $\rho(R_1) = 1 + \frac{0.75}{1} = 1.75$.

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

C. Ranking Score for Routes

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

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

We use $\psi(R)$ for simplicity when the context is clear. Our ranking score can be flexibly customized by the tradeoff parameter $\alpha \in [0,1]$ according to specific application needs [2], [6], [12], [20]. For example, a shopper in a mall may prefer the routes covering the query keywords as much as possible for the sake of shopping, and the requirement for walking distance can be less important. In this case, a large α can boost the keyword score. In contrast, passengers in airports are often more sensitive to distance constraints and would accept some query keywords missing. Thus, a small α can be used to emphasize the distance. The effect of α is studied experimentally.

IV. Search Algorithms for IKRQ

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

A. Pruning Rules for Expansion

Our pruning rules use the skeleton distance [22] as the *lower bound indoor distance* for two indoor items x_i and x_j.

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

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

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

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

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

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

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

Example 7. *Referring to Fig. 1, suppose we need to route from* p_s *to* p_t *with the distance constraint* Δ *= 16m, and we have obtained a partial route* $R^\star = (p_s, d_2, d_5)$ *whose distance is 12.5m.* $R^\star$ *can be pruned according to Pruning Rule 1, since* $\delta(R^\star) + |d_5, p_t|_E$ *= 12.5m + 6m* $> \Delta$ *= 16m. Also, suppose that* $|p_s, d_6|_E$ + $|d_6, p_t|_E$ *= 13m + 5m* $> \Delta$ *= 16m, door* d_6 *can be discarded according to Pruning Rule 2. Take a close look at partition* v_3 *whose only two doors are* d_6 *and* d_7, *and* $|p_s, d_6|_E$ + $|d_6, d_7|_E$ + $|d_7, p_t|_E$ *= 13m + 4.5m + 1m* $> \Delta$ *= 16m,* v_3 *can also be discarded according to Pruning Rule 3.*

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

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

In Pruning Rule 4, we upper bound a partial route's final ranking score by an overestimate of its keyword and spatial scores. The former is overestimated to 1 as a full coverage of query keywords and the latter is computed based on the lower bound indoor distance to p_t (i.e., $\delta(R^\star) + |d_n, p_t|_L$). This pruning rule enables the *k*bound pruning, i.e., we can discard a partial route if its upper bound ranking score is not higher than the k-th best score among the routes already obtained.

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

Lemma 1. *Given a query IKRQ(*p_s, p_t, Δ, *QW, k), if route* $R = (p_s, d_i, \ldots, d_n, p_t)$ *is a returned prime route, each of its partial route* $R^\star = (p_s, \ldots, d_k)$ $(i \le k \le n)$ *is also a prime route.*
Proof. *Without loss of generality, we represent R's remaining route that continues with* $R^\star$ *as* $R^\vdash = (d_k, \ldots, p_t)$. *We prove the lemma by contradiction. Suppose that* $R^\star$ *is not a prime route and* $R^{\star'}$ *is* $R^\star$*'s homogeneous route such that* $\delta(R^{\star'}) < \delta(R^\star)$ *and* $KP(R^{\star'}) = KP(R^\star)$. *According to Definition 2, we have* $R^{\star'}.tail = R^\star.tail = d_k$. *By*

concatenating $R^{\star'}$ and $R^{\vdash}$ at d_k, we can obtain a complete route R' from p_s to p_t. Moreover, we have its sequence of key partitions $KP(R') = concat(KP(R^{\star'}), KP(R^{\vdash})) = concat(KP(R^{\star}), KP(R^{\vdash})) = KP(R)$. According to Definitions 2 and 3, we find R' is a homogeneous route of R and is prime against R. Thus, R cannot be a prime route. □

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

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

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

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

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

Example 8. *Suppose IKRQ(p_s, p_t, 25m, {latte,apple}, 1) is issued in the setting shown in Fig. 1, the parameter α is 0.2, and the search has obtained a complete route $R_1 = (p_s, d_2, d_6, d_7, p_t)$ whose distance is 20m. According to Example 6, we have R_1's keyword relevance is 1.75 and its ranking score is $0.2 \cdot \frac{1.75}{3} + 0.8 \cdot \frac{25-20}{25}$ = 0.277 according to Equation 1. Thus, the current kbound is updated to 0.277. Next, the search expands to a route $R_2^{\star} = (p_s, d_2, d_5, d_7, d_7)$ whose distance is 22.5m and lower bound distance to p_t is 22.5m + $|d_7, p_t|_E$ = 22.5m + 1m = 23.5m. According to Pruning Rule 4, $R_2^{\star}$'s ranking score is upper-bounded by $0.2 \cdot 1 + 0.8 \cdot \frac{25-23.5}{25}$ = 0.248. So $R_2^{\star}$ should be pruned as its upper bound ranking score is smaller than the current kbound.*

Suppose that two partial routes have been obtained between p_s and d_5, namely $R_3^{\star} = (p_s, d_2, d_5)$ and $R_4^{\star} = (p_s, d_3, d_5, d_5)$. Both routes pass a key partition v_2 (its i-word oppo is an indirect matching of apple) and we have $\delta(R_3^{\star})$ = 12.5m and $\delta(R_4^{\star})$ = 23.2m. Currently, $R_3^{\star}$ is prime against $R_4^{\star}$ and therefore $R_4^{\star}$ should be pruned according to Pruning Rule 5. Moreover, according to Lemma 2, when the search has expanded to a door d_9, the next hop cannot be d_9 again as neither of its relevant partitions (v_5 and v_6) covers a query keyword.

B. Overall Search Framework

Based on the above pruning rules and lemmas, we formalize our overall framework in Algorithm 1. A priority queue Q (initialized in line 1) is used to control the order of route expansion. The local information of the current expansion is kept in a five-tuple *stamp* $S(v, R, \delta, \rho, \psi)$, where R is a route that has been expanded to a door or the terminal point so far,

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

```
1:  initialize priority queue Q
2:  set of all candidate i-words W_ci ← ∪_{w_Q∈QW} κ(w_Q).W_i
3:  P ← (∪_{w_Q∈QW} I2P(κ(w_Q).W_i)) \ v(p_s) ∪ v(p_t)
4:  door sets D_n ← ∅, D_f ← ∅
5:  kbound ← 0
6:  initialize hashtable H_prime
7:  R_0 ← (p_s)
8:  S_0 ← (v(p_s), R_0, 0, ρ(R_0), ψ(R_0))
9:  Q.push(S_0)
10: while Q is not empty do
11:     S_i ← Q.pop()
12:     ES ← find(S_i)                      ▷ find the next valid stamps
13:     for each S_j ∈ ES do
14:         connect(S_j)       ▷ connect each valid stamp to terminal
15: return current top-k results
```

v is the last partition that R reaches, and δ, ρ, ψ are R's route distance, keyword relevance, and ranking score, respectively. The architecture of our search algorithms is depicted in Fig. 3.

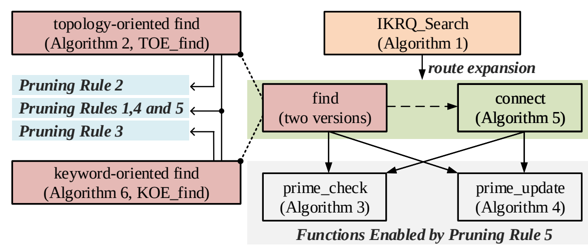

Fig. 3: Architecture of the IKRQ Search Algorithms

The initialization (lines 1–6) obtains a set W_{ci} of all candidate i-words w.r.t. query keyword list QW (line 2), and computes a set P of all key partitions covering at least one keyword in QW (line 3). We exclude the partition $v(p_s)$ from P and add the partition $v(p_t)$ to P to regularize the route search. Sets D_f and D_n hold the doors already explored (line 4). Doors in D_f are filtered by Pruning Rule 2, whereas those in D_n are not. Subsequent routing skips doors in D_f, and exempts doors in D_n from repeated checks by Pruning Rule 2. We initialize the *kbound* for Pruning Rule 4 (line 5), and a hashtable H_{prime} to store the route temporarily prime against others for Pruning Rule 5 (line 6). The algorithm then performs the expansion iteratively (lines 7–14). It generates an initiate route (p_s) and its corresponding stamp S_0 (lines 7–8). Next, it pushes S_0 into Q and iterates on Q until all stamps have been expanded to p_t (lines 9–14). The search follows a *find-and-connect* paradigm. That is, in each iteration, it fetches a stamp S_i with the highest ranking score from Q (line 11), expands the current stamp to find a set ES of valid stamps based on the pruning rules (calling function `find()` in line 12), and attempts to connect each valid stamp in ES to the destination if some condition is met (calling function `connect()` in line 14). The top-k results are returned when Q is empty (lines 10 and 15).

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

C. Topology-oriented Expansion (ToE)

The idea of `find()` in ToE is to reach all accessible doors from the current door based on indoor topology. We formalize this strategy in Algorithm 2. In particular, line 1 initializes a set ES to save the valid stamps to be found, and line 2 obtains the current stamp S_i and the current door d_k from the tail of the corresponding route R_i. To determine if S_i is a temporary prime route that does not need to be pruned (c.f. Pruning Rule 5), ToE calls a function `prime_check()` to compare S_i's route R_i to its homogeneous routes already recorded in a global hashtable H_{prime} (line 3).

Algorithm 2 `ToE_find` (Stamp S_i)

1: set $ES \leftarrow \varnothing$
2: $(v_i, R_i, \delta_i, \rho_i, \psi_i) \leftarrow S_i$; $d_k \leftarrow R_i.tail$
3: **if** `prime_check`(S_i, H_{prime}) is *false* **then return** ▷ Pruning Rule 5
4: **for** each d_l in $P2D_{\sqsubset}(v_i) \setminus D_f$ **do**
5:     **if** $d_l \in R_i$ and $d_l \neq R_i.tail$ **then continue** ▷ regularity check
6:     **if** $d_l \notin D_n$ **then** ▷ Pruning Rule 2
7:         **if** $|p_s, d_l|_L + |d_l, p_t|_L > \Delta$ **then**
8:             $D_f \leftarrow D_f \cup d_l$; **continue**
9:         **else**
10:             $D_n \leftarrow D_n \cup d_l$
11:     $v_j \leftarrow D2P_{\sqsupset}(d_l) \setminus v_i$
12:     **if** $d_k == d_l$ and $PW(v_i).w_i \notin W_{ci}$ **then**
13:         **continue** ▷ regularity check based on Lemma 2
14:     **if** $\delta_i + \delta_{d2d}(d_k, d_l) > \Delta$ **then continue** ▷ distance constraint check
15:     $\delta_{LB} \leftarrow \delta_i + \delta_{d2d}(d_k, d_l) + |d_l, p_t|_L$
16:     **if** $\delta_{LB} > \Delta$ **then continue** ▷ Pruning Rule 1
17:     $\psi_{UB} \leftarrow \alpha \cdot 1 + (1-\alpha)(1 - \delta_{LB}/\Delta)$
18:     **if** $\psi_{UB} \leq kbound$ **then continue** ▷ Pruning Rule 4
19:     $R_j \leftarrow$ append d_l to R_i
20:     $S_j \leftarrow (v_j, R_j, \delta(R_j), \rho(R_j), \psi(R_j))$
21:     `prime_update`(S_j, H_{prime})
22:     add S_j to ES
23: **return** ES

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

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

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

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

Next, ToE performs checks according to the query principles and pruning rules. Particularly, lines 11-13 check the regularity for two identical doors according to Lemma 2, in which v_j is the partition that connects the d_k and d_l on the route. Line 14 checks the distance constraint for the route to be expanded to d_l, and lines 15-16 further derive its lower bound and verify it according to Pruning Rule 1. In the end, ToE uses Pruning Rule 4 to remove the expansion whose derived upper bound ranking score cannot exceed the *kbound* of the search (lines 17-18). Once the check is done, ToE validates the expansion to d_l by appending d_l to the end of R_j and generating the corresponding stamp S_j (lines 19-20). Moreover, it calls function `prime_update()` to update the temporary prime route with S_j (line 19). When each accessible door d_l has been explored, `ToE_find()` returns the set ES that contains all valid stamps.

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

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

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

We proceed to present how to connect each valid stamp returned by `ToE_find()`. The process is formalized in Algorithm 5. For stamp S_j to be connected, we first determine if it has reached the same partition of the terminal point p_t (line 2). If so, we immediately connect the end of the corresponding route R_j to p_t and check if the resulting route R_f meets the query conditions (lines 3–5). If so, we add R_f to the top-k results and update the current *kbound* (lines 6–7). Otherwise, we explore how S_j can be further processed (lines 8–19). Here we call `prime_check()` again to verify if S_j holds the temporary prime route (lines 9–10). Afterwards, we check if the current route R_j has already covered all the query keywords (line 11). If so, there is no necessary to reach any other key partitions. Therefore, we immediately connect the end of R_j to p_t by finding a shortest regular route[4], and obtain a final stamp S_f (lines 12–14). Afterwards, we add the qualified route R_f to the top-k results and update the current *kbound* (lines 15–17). If R_j does not cover all the query keywords, we push S_j into the queue for further expansion (lines 18–19). Lines 2–17 in Algorithm 5 utilize a heuristic rule that the current

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

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

stamp should connect to the destination directly when a certain condition is met, i.e., it has reached the destination partition or covered all query keywords. As a result, the *kbound* and prime routes are updated as soon as possible, which in turn help to prune more aggressively.

Algorithm 5 `connect` (Stamp S_j)

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

D. Keyword-oriented Expansion (KoE)

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

The processing on the current stamp S_i (lines 1–3) is the same as the counterpart in Algorithm 2. It is noteworthy that here v_i must be a key partition and d_k must be an enterable door of v_i since in each expansion KoE has to reach a key partition. Next, unlike ToE that iterates on each enterable door based on indoor topology, KoE searches for the candidate partitions relevant to the uncovered query keywords (lines 4–7). Specifically, it copies the key partition set P (initialized in line 2 of Algorithm 1) to a local set P' (line 4), iterates on each query word $w_Q \in QW$, and checks if w_Q has been covered by the current route R_i in S_i (lines 5–6). If so, its corresponding key partitions should be removed from P' (line 7). In line 6, a case is handled separately. When the initial stamp S_0 is encountered ($d_k = p_s$), we do not remove any partition from P'. This ensures that no extra constraint on partitions is added.

Afterwards, KoE deals with each candidate partition $v_j \in P'$ to find a route that can reach one of the enterable doors of v_j. For each candidate partition v_j, KoE derives the lower bound distance and checks it against Pruning Rule 3 (lines 9–10). If a partition v_j should be pruned, it is excluded from the global set P and never processed in subsequent expansions.

Algorithm 6 `KoE_find` (Stamp S_i)

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

Furthermore, KoE checks the distance constraint for the routes to be expanded to the doors of v_j, whose lower bound distance is computed as $\delta_i + \delta_{LB}(d_k, v_j, p_t)$ (line 11). Referring to Pruning Rule 3, $\delta_{LB}(x_s, v_i, x_t)$ means the minimum indoor distance from x_s, through partition v_i, to x_t.

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

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

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

Lemma 3 can be easily extended to the situation where global regularity needs to be considered for the whole route. Therefore, given any combination of $d_k \in P2D_{\sqsupset}(v_i)$, $d_x \in P2D_{\sqsubset}(v_i)$ and $d_l \in P2D_{\sqsupset}(v_j)$, we only need to find the shortest route $(d_k, d_x, \ldots, d_l)$ with a regularity check (lines 12–13 in Algorithm 6). When each such route has been expanded to d_l, we generate a new route R_j and check it based on Pruning Rule 1 and 4 (lines 14–18). If those rules fail to prune anything, we form a new stamp S_j and call `prime_update()` (lines 19–20). When each candidate partition v_j has been explored, `KoE_find()` returns the set ES that contains all valid stamps. Recall that such stamps will be processed in the search framework (lines 13–14 in Algorithm 1) where the use of `connect()` is the same as that in the search of ToE.

V. Experimental Studies

We experimentally evaluate ToE, KoE and their variants. Table III lists all routing algorithms in comparison. Specifically, ToE\D and KoE\D involve no pruning rule based on the distance constraint Δ, i.e., Pruning Rules 1, 2 and 3. ToE\B and KoE\B skip the *k*bound-based Pruning Rule 4. ToE\P skips the prime-based Pruning Rule 5. This variant does not apply to KoE, since it is formulated based on prime routes. Instead, we design KoE* that precomputes the shortest route between any two doors, which may speed up routing to the next key partition in KoE (line 13 in Algorithm 6). Note that such a route should be re-computed when the regularity check fails. All algorithms are implemented in Java and run on a PC with a 2.30GHz Intel i5 CPU and 16 GB memory.

TABLE III: Notations of Comparable Methods

Modification	**ToE family**	**KoE family**
–	ToE	KoE
no distance-based Pruning Rules 1-3	ToE\D	KoE\D
no kbound-based Pruning Rule 4	ToE\B	KoE\B
no prime-based Pruning Rule 5	ToE\P	–
with precomputed shortest routes	–	KoE*

A. Results on Synthetic Data

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

Indoor Keywords. We assign keywords to the 96 rooms on each floor as follows. We use Scrapy[6] to crawl the online shop information from five shopping malls[7] in Hong Kong, obtaining 2074 documents for 1225 shop brands. All the 1225 brand names are used as i-words. They are then fed into the *RAKE* algorithm [15] to extract corresponding keywords from the documents. Only 1120 i-words yield extracted keywords. For each such i-word, we use up to 60 extracted keywords with the highest TF-IDF values as its t-words. In total, we have 9195 t-words and each i-word corresponds to 16.6 t-words on average. We randomly assign an i-word and all its t-words to each room. The indoor space keyword mappings are of approximately 4 MB and thus kept in main memory.

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

Queries. For a valid IKRQ(p_s, p_t, Δ, QW, k), the distance constraint Δ must be larger than the indoor distance δ_{s2t} between p_s and p_t. Thus, we generate p_s, p_t, and Δ in the following steps. 1) We fix δ_{s2t} to a certain value and randomly select a point p_s in the space. 2) We find a door d' whose distance to p_s approximates δ_{s2t} based on the precomputed door-to-door matrix. 3) We expand from d' to find a random point p_t whose distance to p_s just meets δ_{s2t}. 4) We generate $\Delta = \eta \cdot \delta_{s2t}$, where $\eta > 1$ is a coefficient. Subsequently, we randomly select a set of keywords from the 1120 i-words and 9195 t-words to form QW. A parameter β controls the fraction of i-words in QW. The query keyword set size $|QW|$ is varied from 1 to 5, as an analysis [21] discloses that nearly all map queries contain at most 5 keywords, and statistics[8] show that 65 percent of web searchers use 1 or 2 keywords and over 94 percent of web searchers use at most 4 keywords. In addition, we also vary the tradeoff parameter α in the ranking score (c.f. Equation 1) and the similarity threshold τ (see Definition 4). Table IV gives the parameter settings with default values shown in bold. It is noteworthy that users do not have to specify all of these parameters. For example, users do not need to give i-words and t-words separately. Rather, they are recognized automatically in our implementation.

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

TABLE IV: Parameter Settings

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

2) Efficiency Studies: **Performance Overview.** We run each algorithm in the default setting and report the running time per query instance in Fig. 4. Among all, ToE and KoE perform the best because they make full use of all pruning rules. In general, ToE returns top-7 results within 117ms while KoE needs about 133ms. For ToE\D and KoE\D, the distance-based pruning has a greater impact on their efficiency. Next, ToE\B and KoE\B are basically equal to their original counterparts, showing that the *k*bound pruning barely works in the default setting. The effect of parameter k is studied shortly in the next set of experiments. Still in Fig. 4, the KoE-based algorithms fluctuate more on different query instances than KoE-based ones. This is because the expansion of KoE is highly related to the query words, and thus is easily influenced by the randomly generated QWs. In contrast, ToE's expansion is relatively stable because it always finds the next door according to indoor topology rather than QWs.

KoE* is much slower than others and it has a wider range of variations. This indicates that its precomputing does not pay

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

off. On the contrary, it needs to recompute indoor distances when a route regularity check fails and the recomputed results cannot be reused in a dynamic routing process. Fig. 4 omits the results of ToE\P as it is five to six orders of magnitude slower than the others. ToE\P increases the number of routes exponentially due to its absence of prime route-based pruning. As ToE\P and KoE* perform poorly, we omit them in further comparisons but discuss them separately in Sections V-A3 and V-A4, respectively.

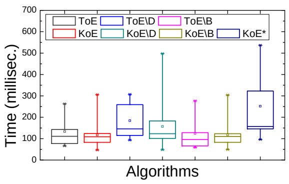

Fig. 4: Default parameters

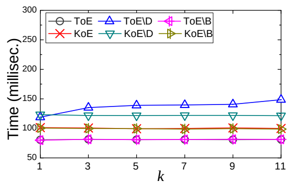

Fig. 5: Running time vs. k

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

Effect of $|QW|$. We vary $|QW|$ from 1 to 5 and report the running time and memory costs in Fig. 6 and 7, respectively. For all algorithms, both metrics increase when $|QW|$ is larger. Referring to Fig. 6, all KoE-based algorithms slow down more rapidly than ToE counterparts. When there are more query words, it is more difficult for partial routes to achieve full coverage of query words and connect to the terminal quickly. Therefore, both ToE and KoE are slower when $|QW|$ increases. Moreover, a larger $|QW|$ leads to more candidate partitions and thus more keyword combinations are considered in KoE. As a result, KoE's running time grows faster than ToE. When $|QW|$ increases to 5, the maximum query keyword size, each KoE-based variant incurs more time than its ToE counterpart. Nevertheless, KoE can still return the top-7 routes within 300ms. Referring to Fig. 7, KoE family cost less memory than ToE family as KoE expansions are more aggressive, jumping directly from one key partition to another without caching intermediate results, whereas KoE has the lowest memory cost thanks to its efficient route pruning.

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

Next, we concentrate on comparing ToE and KoE.

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

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

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

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

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

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

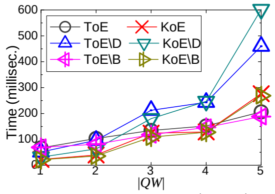

Fig. 6: Time vs. $|QW|$

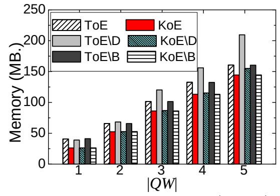

Fig. 7: Memory vs. $|QW|$

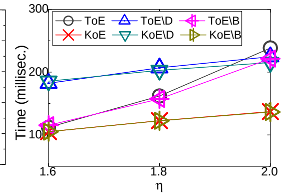

Fig. 8: Time vs. η

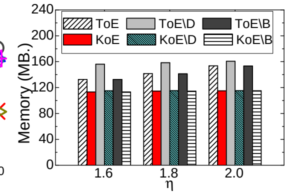

Fig. 9: Memory vs. η

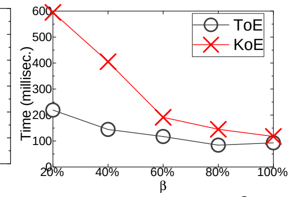

Fig. 10: Time vs. β

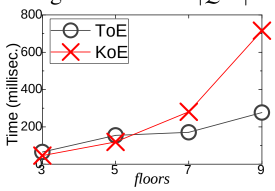

Fig. 11: Time vs. *floor*

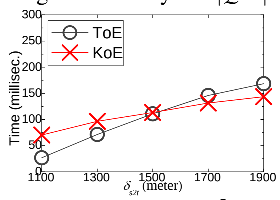

Fig. 12: Time vs. δ_{s2t}

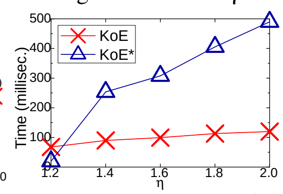

Fig. 13: Time of KoE*

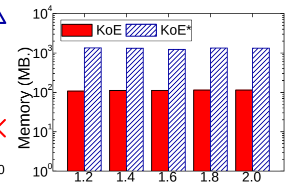

Fig. 14: Memory of KoE*

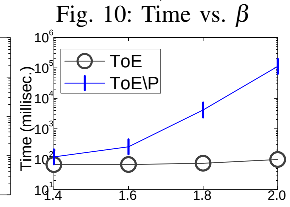

Fig. 15: Time of ToE\P

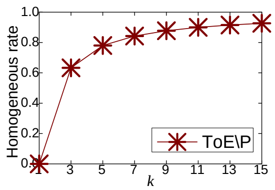

Fig. 16: Homogeneous rate

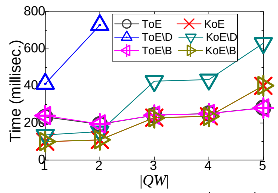

Fig. 17: Time vs. $|QW|$

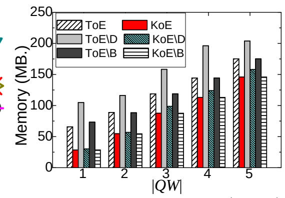

Fig. 18: Memory vs. $|QW|$

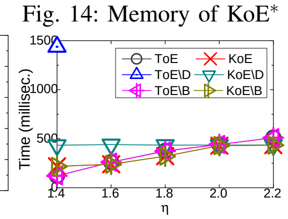

Fig. 19: Time vs. η

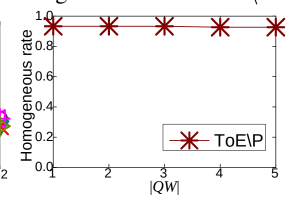

Fig. 20: Homogeneous rate

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

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

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

5) Search Result Quality: We use a typical example to show that our IKRQ can find more reasonable and desirable routes in practice. Referring to Fig. 1, we have $I2T(Apple)$ = $\{phone, mac, laptop, watch\}$ and $I2T(Samsung)$ = $\{phone, laptop, earphone\}$. Assuming α is 0.5 and τ is 0.1, query $(p_1, p_2, 100, earphone, 2)$ returns routes R_1 = $(p_1, d_4, d_{15}, d_{15}, p_2)$ and R_2 = $(p_1, d_4, d_{17}, d_{17}, p_2)$, although *earphone* is not in Apple's t-words in R_1. In particular, $\delta(R_1)$ = $10m$, $\delta(R_2) = 20m$, $\rho(R_1) = 1.667$ and $\rho(R_2) = 2$, so $\psi(R_1)$ = 0.867 and $\psi(R_2)$ = 0.9. Although $R_3 = (p_1, d_4, p_2)$ has a shorter distance of $9.5m$, it lacks words similar with *earphone* and is not returned with $\psi(R_3) = 0.4525$. Apparently, Apple offers earphones. Its route R_1 will be excluded and users will miss useful choices if we use exact keyword matching.

B. Results on Real Data

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

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

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

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

VI. Related Work

Indoor Routing and Path Finding. Goetz and Zipf [5] define a routing graph for indoor environments with obstacles. Lu et al. [13] design an indoor space model that facilitates shortest path finding. To speed up distance-aware indoor path finding, Shao et al. [17] design VIP-tree that enables more aggressive pruning. VIP-tree also supports indoor trip planning based on neighbour expansion [18]. Li et al. [10] construct indoor possible paths based on probabilistic location samples of moving objects and search for the most popular indoor semantic regions using the constructed paths. Costa et al. [3] propose context-aware indoor-outdoor path recommendation that minimizes the outdoor exposure and path distance. Li et al. [8] design vision-based mobile indoor navigation that helps blind and visually impaired people walk indoors. In contrast to our IKRQ, these works do not consider indoor semantic keywords. A recent work [16] studies indoor keyword-aware skyline route query that considers the number of covered keywords and route distances, whereas our IKRQ does not count keywords but use prime routes to exclude routes through the same partitions. Also, unlike work [16], our setting allows a partition to have more than one keyword.

Outdoor Keyword-aware Routing. Given a source s, a destination e, and a category set C, the trip planning query [9] finds the shortest s-to-e path that covers at least one object from each category in C, whereas the optimal sequenced route query [19] finds the shortest path covering all categories in a total order. Partial order is considered elsewhere [11]. The multi-approximate-keyword routing query [23] changes the strict category coverage to an approximate matching using edit distances between a keyword and a location. The geographical route search [7] finds routes whose length is within a threshold and keyword-dependent scores are highest. The keyword-aware optimal routing [1] considers keyword coverage, route score, and travel cost budget. The optimal route search [24] finds one route whose word coverage is maximum within a budget constraint. The clue-based route search [25] supports an order of keywords to cover, and requires that the network distance from one matched keyword to next is within a corresponding user-specified limit. However, all these works fall short for indoor topology considered in our IKRQ queries. Also, none of them distinguishes identity and content words that carry different semantics. Moreover, most works do not consider routing diversity, and works [1], [7], [9], [23], [24] are approximate solutions.

VII. Conclusion and Future Work

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

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

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

References

[1] X. Cao, L. Chen, G. Cong, and X. Xiao. Keyword-aware optimal route search. *PVLDB*, 5(11):1136–1147, 2012.
[2] G. Cong, C. S. Jensen, and D. Wu. Efficient retrieval of the top-k most relevant spatial web objects. *PVLDB*, 2(1):337–348, 2009.
[3] C. Costa, X. Ge, and P. Chrysanthis. CAPRIO: Context-Aware Path Recommendation Exploiting Indoor and Outdoor Information. *MDM*, 431–436, 2019.
[4] G. J. Fakas, Y. Cai, Z. Cai, and N. Mamoulis. Thematic ranking of object summaries for keyword search. *Data Knowl. Eng.*, 1–17, 2018.
[5] G. 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.
[6] R. Hariharan, B. Hore, C. Li, and S. Mehrotra. Processing spatial-keyword (SK) queries in geographic information retrieval (GIR) systems. *SSDBM*, page 16, 2007.
[7] Y. Kanza, E. Safra, Y. Sagiv, and Y. Doytsher. Heuristic algorithms for route-search queries over geographical data. In *ACM-GIS*, 11, 2008.
[8] B. Li, J. P. Muñoz, X. Rong, Q. Chen, J. Xiao, Y. Tian, A. Arditi, and M. Yousuf. Vision-based mobile indoor assistive navigation aid for blind people. *TMC*, 18(3):702–714, 2018.
[9] F. Li, D. Cheng, M. Hadjieleftheriou, G. Kollios and S-H. Teng. On Trip Planning Queries in Spatial Databases. In *SSTD*, pages 273–290, 2005.
[10] H. Li, H. Lu, L. Shou, G. Chen, and K. Chen. Finding Most Popular Indoor Semantic Locations Using Uncertain Mobility Data. *TKDE*, 31(11):2108–2123, 2018.
[11] J. Li, Y. D. Yang, and N. Mamoulis. Optimal route queries with arbitrary order constraints. *TKDE*, 25(5):1097–1110, 2013.
[12] Z. Li, K. C. K. Lee, B. Zheng, W.-C. Lee, D. L. Lee, and X. Wang. IR-tree: An efficient index for geographic document search. *TKDE*, 23(4):585–599, 2010.
[13] H. Lu, X. Cao, and C. S. Jensen. A foundation for efficient indoor distance-aware query processing. In *ICDE*, pages 438–449, 2012.
[14] L. Qin, J. X. Yu, and L. Chang. Diversifying top-k results. *PVLDB*, 5(11):1124–1135, 2012.
[15] S. Rose, D. Engel, N. Cramer, and W. Cowley. Automatic keyword extraction from individual documents. *Text mining: applications and theory*, Wiley, 3–20, 2010.
[16] C. Salgado. Keyword-aware Skyline Routes Search in Indoor Venues. *SIGSPATIAL ISA*, 25–31, 2018.
[17] 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.
[18] Z. Shao, M. A. Cheema, and D. Tania. Trip planning queries in indoor venues. *The Computer Journal*, 61(3):409–426, 2017.
[19] M. Sharifzadeh, M. Kolahdouzan, and C. Shahabi. The optimal sequenced route query. *VLDBJ*, 17(4):765–787, 2008.
[20] D. Wu, G. Cong, and C. S. Jensen. A framework for efficient spatial web object retrieval. *VLDBJ*, 21(6):797–822, 2012.
[21] X. Xiao, Q. Luo, Z. Li, X. Xie and W.-Y. Ma. A large-scale study on map search logs. *TWEB*, 4(3):1-33, 2010.
[22] X. Xie, H. Lu, and T. B. Pedersen. Efficient distance-aware query evaluation on indoor moving objects. In *ICDE*, pages 434–445, 2013.
[23] B. Yao, M. Tang, and F. Li. Multi-approximate-keyword routing in GIS data. In *ACM-GIS*, pages 201–210, 2011.
[24] Y. Zeng, X. Chen, X. Cao, S. Qin, M. Cavazza and Y. Xiang. Optimal route search with the coverage of users' preferences. In *IJCAI*, pages 2118–2124, 2015.
[25] B. Zheng, H. Su, W. Hua, K. Zheng, X. Zhou and G. Li. Efficient clue-based route search on road networks. *TKDE*, 29(9): 1846–1859, 2017.