Time-Constrained Indoor Keyword-aware Routing

Harry Kai-Ho Chan*, Tiantian Liu†, Huan Li†, Hua Lu*
*Department of People and Technology, Roskilde University, Denmark
†Department of Computer Science, Aalborg University, Denmark
*{kai-ho, luhua}@ruc.dk †{liutt, lihuan}@cs.aau.dk

ABSTRACT

With the increasingly available indoor positioning technologies, indoor location-based services (LBS) are becoming popular. Among indoor LBS applications, indoor routing is particularly in demand. In the literature, there are several existing studies on indoor keyword-aware routing queries, each considering different criteria when finding an optimal route. However, none of these studies explicitly constraint the time budget for the route. In this paper, we propose a new problem formulation TIKRQ that considers the time needed for a user to complete the route, in addition to other criteria such as static cost and textual relevance. A set-based search algorithm and effective pruning strategies are proposed for TIKRQ. We conduct extensive experiments to verify the efficiency of our proposals.

CCS CONCEPTS

• **Information systems → Location based services**.

KEYWORDS

indoor space, indoor query processing, keyword-aware, routing

ACM Reference Format:
Harry Kai-Ho Chan, Tiantian Liu, Huan Li, and Hua Lu. 2021. Time-Constrained Indoor Keyword-aware Routing. In *17th International Symposium on Spatial and Temporal Databases (SSTD '21), August 23–25, 2021, virtual, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3469830.3470895

1 INTRODUCTION

With recent developments in indoor positioning technologies and the widespread use of smartphones, indoor location-based services (LBS) [1] are becoming increasingly popular. Typical indoor LBS related applications include finding interested indoor objects and locations [13, 27, 30, 32, 33], indoor navigation and route planning [9, 16, 23–25], and indoor movement pattern mining [12, 14]. Among them, indoor route planning is particularly in demand, which assists users in planning a route satisfying their preferences, especially in an unfamiliar and large indoor environment like an airport or a shopping mall.

Consider that Alice has just passed the security check in the airport. As she has 90 minutes before the boarding time of the flight, she wants to buy a coffee, some souvenirs and a new charging cable. She can issue an indoor routing query from a source point (her current location) to a target point (i.e., the boarding gate), and specify the preferences by some keywords (e.g., coffee, souvenir, and charging cable). The query should return a route that passes through a shop that sells coffee, a souvenir shop, and a shop that sells charging cables. Most importantly, she should be able to complete the route within the 90-minute time constraint.

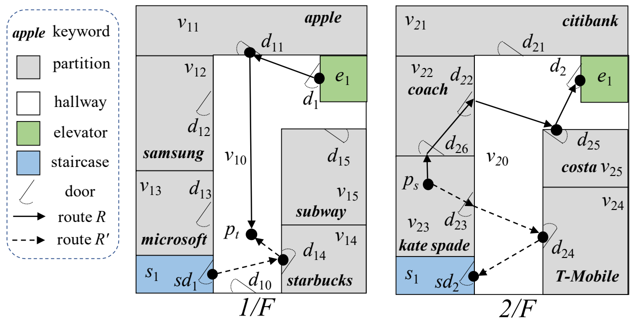
There are several existing studies on indoor keyword-aware routing query [9, 23–25], each of which considers different criteria when finding the result, including route distance, keyword relevance, and static cost. These existing route planning queries focus on minimizing the total length of the route that visits all requested keywords. However, none of these studies considers the *time constraint* as a hard constraint. In this paper, we propose a new problem formulation that is capable of taking all these criteria into account.

In practice, measuring the time needed to complete a route is much more reasonable and comprehensive than focusing on the route distance. First, instead of giving a concrete maximum walking distance (e.g., 500m), a user might feel more friendly to give the time she/he is willing to walk (e.g., 5 minutes), especially in some time-sensitive situations (e.g., as in our example, the user has to arrive at the boarding gate before a particular boarding time). Second, the distance metric overlooks the distances we travel by other means, such as elevators in the shopping malls and Automated People Mover (APM) in the airports. For example, the APM in Hong Kong International Airport needs 10 minutes to arrive at the farthest midfield concourse [1]. While these travel means do not incur any walking distance, the time needed on them should not be simply neglected when planning the route. A recent work [9] converts a time constraint into a distance constraint by multiplying the former by a maximum indoor walking speed, which, however, cannot handle these cases properly. Third, the waiting time of the shops (e.g., queuing time for a restaurant, or checkout time needed for a supermarket) should also be taken into account when returning a route to the user.

Following a previous work [24], we also consider the static cost of shops in this paper. The static cost of a shop could refer to the average price of the products in the shop, or an estimated crowdedness of the shop. It could also be a composite cost of different criteria, such as price, popularity, and rating.

In this paper, we propose a new problem formulation that takes the route's *time needed* into account, rather than the (walking) distance. Specifically, we formulate a time-constrained indoor keyword-aware routing query (TIKRQ). A TIKRQ requires a source point p_s, a target point p_t, a set QW of query keywords and a time constraint Δ_{Max}. It returns the top-k routes from p_s to p_t such that each of their total time needed is less than Δ_{Max} and their costs

[1]https://en.wikipedia.org/wiki/Hong_Kong_International_Airport_Automated_People_Mover



Partition	Shop	Static Cost	Waiting Time (min.)
v_{11}	apple	6	8
v_{25}	costa	2	4
v_{14}	starbucks	3	3
v_{24}	T-mobile	8	5
...	...	...	...

Figure 1: Running Example

are the minimum compared to those of the others. In our setting, the cost of a route captures the static cost and the textual relevance of the indoor partitions with respect to QW.

Figure 1 shows our running example, which consists of a floor plan of a shopping mall with two floors, and a table listing the shops' static costs and waiting time. Suppose a user will meet her friends in 20 minutes. During her spare time, she wants to buy a coffee and a charging cable for her phone. She then issues a query with her current location as the source point p_s (in v_{23} on 2/F), the meeting location as the target point p_t (in v_{10} on 1/F), two query keywords *coffee* and *charging cable*, and a route's time constraint of 20 minutes. Both routes R and R' are possible solutions of the query. Route R visits *costa* (which is a coffee shop) and *apple* (which sells charging cables), and uses the elevator e_1 to reach 1/F from 2/F, while route R' visits *starbucks* (which is another coffee shop) and *T-Mobile* (which also sells charging cables) and goes from 2/F to 1/F by the staircase s_1. Suppose the total walking time of R and R' is 5 and 4 minutes, respectively, and the elevator takes 1 minute to go from 2/F to 1/F. The total time needed to complete R and R' is $5 + 1 + 8 + 4 = 18$ minutes and $4 + 5 + 3 = 12$ minutes, respectively. In this case, both R and R' can be completed within 20 minutes, while R should be more desirable to the user, since R has a total static cost of $2 + 6 = 8$ which is much smaller than that $8 + 3 = 11$ of R'.

In addition, we develop a concept of *unique partition set* to improve the diversity of the top-k results. Also, we organize and distinguish three types of indoor keywords to better capture the semantics of the query keywords. To answer the TIKRQ, we propose an algorithm based on a novel set-based search strategy to search for the top-k routes. Efficient pruning techniques and computation strategies are also designed to improve performance.

Compared to the existing studies [9, 23–25], our proposed TIKRQ (1) provides the flexibility for users to specify the *time constraint* of the routes, (2) is more comprehensive as it also considers the static cost of the partitions, (3) better organizes the keyword in an indoor setting by introducing an additional *category-word*, and (4) adopts the concept of unique partition set to improve the diversity of the top-k results.

The contributions of this work are summarized as follows.

- We formulate the time-constrained indoor keyword-aware routing query (TIKRQ) that takes the routes' time needed into consideration. We also propose a concept of unique partition set to diversify the top-k routes. (Section 2)
- We propose a keyword organization for indoor keywords, and a method to compute textual relevance for routes. (Section 3)
- We design a set-based search algorithm with effective pruning techniques to resolve TIKRQ. (Section 4)
- We conduct extensive experiments and case studies to evaluate the proposed techniques. (Section 5)

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

2 PROBLEM DEFINITION

2.1 Preliminaries

Table 1 shows the frequently used notations in this paper.

Table 1: Notations

Notation	Description
v, d, p	Partition, door, and point in an indoor space
$v.cost$, $v.waitTime$	Static cost and waiting time of a partition v
e	Transport in indoor space (e.g., an elevator)
w	A word in a partition
QW	The set of query keywords
$KPS(R)$	The set of key partitions on route R
$PC(R)$	Partition cost of route R
$\gamma(R)$	Time cost of route R
$\rho(R)$	Textual relevance of route R
$cost(R)$	Route cost of route R
CKP	A set of candidate key partitions
S	A key partition set

A partition and a transport are the basic building blocks in an indoor space. A point is located inside a partition or a transport. In an indoor routing, one needs to move from one door to another through their common partition or transport. Following [17], we use the mapping to capture the indoor topology: Given a door d_i, we use $D2P_{\sqsupset}(d_i)$ and $D2P_{\sqsubset}(d_i)$ to denote the set of partitions and transports that one can enter and leave through d_i, respectively.

Given a partition v_k or a transport e_k, and two different doors d_i and d_j, the intra-partition door-to-door distance from d_i to d_j is defined as

$$\delta_{d2d}(d_i, d_j) = \begin{cases} |d_i, d_j|_E, & \text{if } v_k \in D2P_{\sqsupset}(d_i) \text{ and } v_k \in D2P_{\sqsubset}(d_j) \\ 0, & \text{if } e_k \in D2P_{\sqsupset}(d_i) \text{ and } e_k \in D2P_{\sqsubset}(d_j) \\ \infty, & \text{otherwise} \end{cases}$$

In the case that d_i and d_j are in the same partition (say v_k), one can enter the partition from d_i and leave by d_j. We measure the distance between d_i and d_j by Euclidean distance. Other distance metrics such as obstacle distance can also be adopted here. Following [9], we handle the special case of $d_i = d_j$, which happens when one needs to enter a partition due to its keyword relevance but then leave it from the same door for further routing, as follows. We set $\delta_{d2d}(d_i, d_i)$ to be the double of the longest non-loop distance one can reach inside the partition from the door d_i. In the case that d_i and d_j are in the same transport, we simply regard the door-to-door distance from d_i to d_j as zero. To reflect the practical time needed to pass a transport, we assign each transport a waiting time, as to be detailed in Section 3.1.

Given a point p_i, we use $v(p_i)$ to denote the partition or transport that contains p_i. Given a partition v_i, we use $P2D_{\sqsupset}(v_i)$ and $P2D_{\sqsubset}(v_i)$ to denote the set of doors through which one can enter and leave the partition v_i, respectively. Similarly, given a transport e_i, we use $E2D_{\sqsupset}(e_i)$ and $E2D_{\sqsubset}(e_i)$ to denote the set of doors which one can enter and leave the transport e_i, respectively. Given a door d_k and a point p_i, the point-to-door distance and door-to-point distance are defined as

$$\delta_{pt2d}(p_i, d_k) = \begin{cases} |p_i, d_k|_E, & \text{if } d_k \in P2D_{\sqsubset}(v(p_i)) \\ 0, & \text{if } d_k \in E2D_{\sqsubset}(v(p_i)) \\ \infty, & \text{otherwise} \end{cases}$$

$$\delta_{d2pt}(d_k, p_i) = \begin{cases} |d_k, p_i|_E, & \text{if } d_k \in P2D_{\sqsupset}(v(p_i)) \\ 0, & \text{if } d_k \in E2D_{\sqsupset}(v(p_i)) \\ \infty, & \text{otherwise} \end{cases}$$

When the context is clear, we use $\delta_*(x_i, x_j)$ to indicate the distance from a point/door x_i to a point/door x_j.

2.2 Problem Definition

Definition 1 (Route [9]). *A **route** $R = (x_s, d_i, ..., d_n, x_t)$ is a path through a sequence of doors from point/door x_s to x_t. A route is a **complete route** if x_s and x_t are the source and target points, respectively. Otherwise, it is a **partial route**.*

We can easily obtain the partitions and transports that a route R passes using the aforementioned indoor topological mappings.

Consider the partial route R on 2/F in Figure 1, we have $R = (p_s, d_{26}, d_{22}, d_{25}, d_{25}, d_2)$. We know that R passes v_{23}, v_{22}, v_{25} and v_{20} since $R = (p_s \xrightarrow{v_{23}} d_{26} \xrightarrow{v_{22}} d_{22} \xrightarrow{v_{20}} d_{25} \xrightarrow{v_{25}} d_{25} \xrightarrow{v_{20}} d_2)$.

We use the term **relevant partition** [9] to refer to a partition that covers p_s, p_t or a subset of query keywords. Given a route R, a **key partition** of R is a partition that R has been through and that has the maximum keyword relevance (to be given in Definition 6) for at least one query keyword. We use $KPS(R)$ to denote the set of key partitions in R. Considering the example in Figure 1 with query keywords *starbucks* and *apple*, partitions v_{10}, v_{11}, v_{14}, and v_{23} are relevant partitions and $KPS(R) = \{v_{11}, v_{14}\}$.

Definition 2 (Partition Cost). *We define the partition cost of a route as the sum of the static cost of its key partitions.*

$$PC(R) = \sum\nolimits_{v \in KPS(R)} v.cost$$

Note that any monotonic function can be used to model a route's partition cost. In this paper, we use the sum function for conciseness.

Definition 3 (Route Cost). *We define the cost of a route as the linear combination of its partition cost and textual relevance, i.e.,*

$$cost(R) = \alpha \cdot \frac{PC(R)}{PC_{max} \cdot |QW|} + (1 - \alpha) \cdot (1 - \rho(R)) \qquad (1)$$

where $\alpha \in [0, 1]$ is a user parameter, PC_{max} is the maximum partition cost in the indoor venue, and $\rho(R)$ is the textual relevance of R (to be defined in Section 3.2).

The parameter α controls the weighting between the partition cost and textual relevance, and can be tuned according to the user needs. A smaller α puts a larger weight on route's textual relevance, while a larger α focuses more on the partition cost. We vary and evaluate this parameter in Section 5.2.1.

We define our problem as follows.

Problem 1 (Time-Constrained Indoor Keyword-aware Routing). *Given a source point p_s, a target point p_t, a set QW of query keywords, a time constraint Δ_{Max} and an integer k, a* Time-Constrained Indoor Keyword-aware Routing Query *$TIKRQ(p_s, p_t, QW, \Delta_{Max}, k)$ returns k complete routes with the smallest route cost, and each such a route R from p_s to p_t (i.e., $R = (p_s, ..., p_t)$) has a time cost $\gamma(R)$ less than the time constraint (i.e., $\gamma(R) < \Delta_{Max}$).* □

Above, $\gamma(R)$ captures the time needed for R to complete the given routing query (to be defined in Section 3.1). We say that a route is a **feasible route** if it satisfies the time constraint. Therefore, the TIKRQ is to find k feasible routes with minimum cost.

To ensure the resulting routes are meaningful, we use the following two principles of indoor routing [9].

Principle of Regularity. Unlike traditional outdoor routing algorithms [2, 34] that exclude loops in a route to avoid endless route searching, we allow a regular route in the indoor space to have a loop of doors in some cases. Consider the example in Figure 1, a user who wants to visit partition v_{14} must enter and leave d_{14}, producing a partial route $(..., d_{14}, d_{14}, ...)$. The principle of regularity disqualifies a route that contains a loop without any key partitions in the loop. That is, we exclude loops in a route between any two key partitions. For example, for a query with the keyword *starbucks*, $R' = (p_s, d_{26}, d_{26}, d_{23}, sd_2, sd_1, d_{14}, d_{14}, p_t)$ is not allowed since v_{22} visited by the loop (d_{26}, d_{26}) is not a key partition of the query.

Principle of Diversity. The concepts of diversifying top-k results [20, 35] and prime route [9] inspire us to avoid homogeneous routes in our routing results. We propose a concept of **unique partition set**. Specifically, for each of the k resulting routes, its key partition set must be unique. That is, for any two resulting routes R and R', we must have $KPS(R) \neq KPS(R')$.

Table 2: Four Example Routes from p_s to d_2 Each Covering *costa* and *citibank*

R_1	$(p_s \xrightarrow{v_{23}} d_{26} \xrightarrow{v_{22}} d_{22} \xrightarrow{v_{20}} d_{21} \xrightarrow{v_{21}} d_{21} \xrightarrow{v_{20}} d_{25} \xrightarrow{v_{25}} d_{25} \xrightarrow{v_{20}} d_2)$
R_2	$(p_s \xrightarrow{v_{23}} d_{26} \xrightarrow{v_{22}} d_{22} \xrightarrow{v_{20}} d_{25} \xrightarrow{v_{25}} d_{25} \xrightarrow{v_{20}} d_{21} \xrightarrow{v_{21}} d_{21} \xrightarrow{v_{20}} d_2)$
R_3	$(p_s \xrightarrow{v_{23}} d_{23} \xrightarrow{v_{20}} d_{25} \xrightarrow{v_{25}} d_{25} \xrightarrow{v_{20}} d_{21} \xrightarrow{v_{21}} d_{21} \xrightarrow{v_{20}} d_2)$
R_4	$(p_s \xrightarrow{v_{23}} d_{23} \xrightarrow{v_{20}} d_{21} \xrightarrow{v_{21}} d_{21} \xrightarrow{v_{20}} d_{25} \xrightarrow{v_{25}} d_{25} \xrightarrow{v_{20}} d_2)$

Consider Figure 1 as an example. Suppose a user wants routes from p_s to d_2 while covering two keywords $QW = \{costa, citibank\}$ in the route. Several possible routes are listed in Table 2. For ease of illustration, we insert the partitions that connect two consecutive items in the route. We can see that $KPS(R_1)$=$KPS(R_2)$=$KPS(R_3)$ = $KPS(R_4)$= $\{v_{23}, v_{21}, v_{25}, v_{20}\}$. The four routes pass the same set of key partitions with different orders and different partial routes in-between. Thus, only one of the four routes should be included in the query result.

Note that this requirement provides a more diversified result than a prime route [9], as the unique partition set is more *restrictive* than the prime route. In particular, the concept of prime route only requires $SRP(R) \neq SRP(R')$, where $SRP(R)$ denotes the *sequence* of relevant partitions in R. In our example, R_1 and R_2 (or R_3 and R_4) could be in the prime route query result at the same time, since $SRP(R_1)=\langle v_{23}, v_{21}, v_{25}, v_{20}\rangle$ is different from $SRP(R_2) = \langle v_{23}, v_{25}, v_{21}, v_{20}\rangle$.

3 TIME COST AND TEXTUAL RELEVANCE

In this section, we detail the formulation of the time cost $\gamma(R)$ in Section 3.1 and the textual relevance $\rho(R)$ in Section 3.2.

3.1 Time Cost

In this paper, we consider two types of time for a route R.

Travelling Time. The travelling time of a route R, denoted by $t_{travel}(R)$, refers to the time needed for a user to complete R. As discussed in Section 1, both walking and taking a transport incur travelling time, and we model them as follows. Assuming the average human walking speed s_{walk} is 5km/hour [2], we can easily compute the travelling time on walking as the walking distance divided by the walking speed.

To model the travelling time on taking transport, consider a user taking an elevator for illustration. To take an elevator, the total journey time includes waiting (outside the elevator) and travelling (inside the elevator). The estimation of this waiting time can be based on the average value of previous records, which is beyond the scope of this paper. This paper assumes a fixed waiting time (e.g., 30 seconds), rather than a distribution, for ease of illustration. Similar to computing the walking time, the travelling time on an elevator is the height of travel from one floor to another divided by the elevator's speed.

Based on the above, given two doors d_i and d_j that connect to a partition v or a transport e, the time needed to travel from d_i to d_j is defined as

$$\gamma(d_i, d_j) = \begin{cases} \frac{\delta_{d2d}(d_i, d_j)}{s_{walk}}, & \text{if } v \in D2P_{\sqsupset}(d_i) \text{ and } v \in D2P_{\sqsubset}(d_j) \\ \frac{|d_i, d_j|}{s_e} + e_{wait}, & \text{if } e \in D2P_{\sqsupset}(d_i) \text{ and } e \in D2P_{\sqsubset}(d_j) \\ \infty, & \text{otherwise} \end{cases}$$

where $|d_i, d_j|$ is the actual distance of the two doors in e, s_e is the moving speed of the transport, and e_{wait} is the waiting time of the transport. For simplicity, we assume that the start point and target point are located in partitions only [3].

The travelling time of $R = (p_s, d_i, \ldots, d_k, p_t)$ can be computed as follows.

$$t_{travel}(R) = \frac{\delta_*(p_s, d_i)}{s_{walk}} + \sum_{k=i}^{n-1} \gamma(d_k, d_{k+1}) + \frac{\delta_*(d_n, p_t)}{s_{walk}}$$

[2]We use a universal walking speed in this paper for ease of illustration, but the proposed method can be easily adapted to the walking speed tailored for partitions.

[3]We do not consider the extreme case that the source and target points are located in the transport, but our technique can easily support it.

Partition Time. The partition time of a route R, denoted by $t_{part}(R)$, is the sum of time spent in the key partitions where the user stays to fulfill her purposes implied by the keywords, i.e.,

$$t_{part}(R) = \sum\nolimits_{v \in KPS(R)} v.waitTime$$

where $v.waitTime$ denotes the waiting time of the partition v. Similar to the transport's waiting time, we assume a fixed value for the waiting time in each partition.

Time Cost. Based on the above, we define the time cost of a route by the following cost function.

Definition 4 (Time Cost). *Given a route R, the time cost of R, denoted by $\gamma(R)$, is defined as the sum of the travelling time and the waiting time of R.*

$$\gamma(R) = t_{travel}(R) + t_{part}(R) \qquad (2)$$

3.2 Textual Relevance

Keywords in Indoor Space. In the literature, an **identity word** (i-word) [9] identifies the specific name of a partition (e.g., *starbucks, apple*), and a **thematic word** (t-word) [8, 9] refers to a tag relevant to that partition (e.g., *coffee, laptop*). In addition, we employ a **category word** (c-word) that specifies the type of the partition (e.g., *coffee shop, supermarket*). A partition can be associated with one c-word and one i-word, but a set of t-words. For example, a partition in a shopping mall is associated with a c-word *coffee shop*, an i-word *starbucks* and t-words *coffee, mocha, latte*; another partition can be associated with a c-word *electronics*, an i-word *apple*, and t-words *smartphone, laptop, headphone.* Note that it is possible to extend our organization to support one partition associated with multiple or hierarchical c-words, which is left for future work.

Insufficiency of Existing Setting. The previous work [9] differentiates two types of keywords associated with indoor partitions. In particular, they assumed that two partitions having the same i-word *must* have the same set of t-words. However, this assumption over-simplifies the case. For example, depending on the shop's size and location, two starbucks can have different menus. A smaller one might not have some products (e.g., cakes and juices) for sale, while the one close to a train station sells more grab-and-go foods. As another example, some ATMs offer different currencies in cash, while some others do not. Compared to the assumption and the limitations in the organization of indoor space keywords in [9], our keyword organization, which we introduce below, is more general and comprehensive.

We assume that the three sets of words are disjoint for ease of illustration. Given a partition v_i, a **P2I** mapping $P2I(v_i)$ maps v_i to its associated i-word, and a **P2T** mapping $P2T(v_i)$ maps v_i to its associated t-words. Given an i-word w_i, an **I2C** mapping $I2C(w_i)$ maps w_i to its associated c-word, and an **I2P** mapping $I2P(w_i)$ maps w_i to the partitions associated with it. Given a t-word w_t, a **T2P** mapping $T2P(w_t)$ maps w_t to the partitions associated with it. Given a c-word w_c, a **C2I** mapping $C2I(w_c)$ maps w_c to the associated i-words.

To better represent the real-world setting, we maintain P2I as a many-to-one mapping and I2P as a one-to-many mapping such that a partition can be associated with one i-word, and each i-word can be associated with multiple partitions. For example, there could be multiple *starbucks* in a mall. We maintain P2T and T2P as two many-to-many mappings, meaning that each partition can be associated with multiple t-words and vice versa. Besides, we maintain I2C as a many-to-one mapping and C2I as a one-to-many mapping. Figure 2 shows an example of the organization of indoor space keywords.

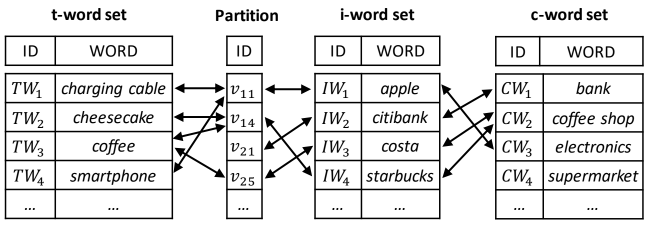


Figure 2: Keyword Mappings in Indoor Space

Given the organization described above, we are now ready to introduce the calculation of keyword relevance between the query keywords and a route as follows.

Keyword Relevance Computation. Given a set QW of query keywords, we first match each query word $w \in QW$ to the **candidate partitions** for facilitating the routing afterwards.

Definition 5 (Candidate Partitions). *Given a query keyword $w \in QW$, its candidate partitions $\mathcal{CP}(w)$ is represented as a set of entries each of which is in the form of (v_i, rs), where v_i is the matching partition and rs is the relevance score between v_i and w. We discuss different cases based on the type of w as follows.*

- *Case 1 (w is a c-word): All partitions associated with the matching i-words in $C2I(w)$ are matched with $rs = 1$.*
- *Case 2 (w is an i-word): All partitions associated with the i-word w are matched with $rs = 1$. To enrich the result, we also include partitions associated with other i-words. In particular, all partitions associated with an i-word w'_i such that $I2C(w'_i) = I2C(w)$ are matched with $rs = 0.1$* [4].
- *Case 3 (w is a t-word): All partitions associated with the matching t-word w are matched with $rs = 1$. All partitions v_j associated with t-words t'_i such that $t'_i \in \bigcup_{v_i \in T2P(w)} P2T(v_i)$ are matched with $rs = \frac{P2T(v_j) \cap \bigcup_{v_i \in T2P(w)} P2T(v_i)}{P2T(v_j) \cup \bigcup_{v_i \in T2P(w)} P2T(v_i)}$ based on the Jaccard Similarity.*

Compared to [9], our definition has an extra case that w is a c-word, and it uses a different scoring scheme to handle the case that w is an i-word, which is designed based on our new keyword organization.

Definition 6 (Keyword Relevance). *Given a route R and a query keyword w_Q, we define the keyword relevance of w_Q w.r.t. R as the maximum rs of $v_i \in R$ as follows.*

$$rel(R, w_Q) = \max_{(v_i, rs) \in \mathcal{CP}(w_Q) | v_i \in R} \mathcal{CP}(w_Q).rs$$

Definition 7 (Textual Relevance). *Given a route R, we define the textual relevance $\rho(R)$ as the sum of keyword relevance of all query keywords as follows.*

$$\rho(R) = \Big(\sum_{w_Q \in QW} rel(R, w_Q)\Big) / |QW|$$

where $|QW|$ is the normalization term to make $\rho(R)$ fall in $[0, 1]$.

[4]Any small value can be used here as long as the original i-word w has a higher score. The routes with w will have higher rankings than those with w'_i.

Consider our example in Figures 1 and 2. Suppose the query keywords are *coffee* and *charging cable* (both keywords are t-words). R passes the key partitions v_{11} and v_{25}, which is associated with *charging cable* (i.e., $v_{11} \in T2P(TW_1)$) and *coffee* (i.e., $v_{25} \in T2P(TW_3)$), respectively. Thus, we have $\rho(R) = \frac{1+1}{2} = 1$. If the query keywords are changed to *starbucks* (which is an i-word) and *electronics* (which is a c-word), v_{11} and v_{25} are still the key partitions of R, and we have $\rho(R) = \frac{0.1+1}{2} = 0.55$ since $I2P(v_{11}) =$ *costa* is of the same category *coffee shop* with *starbucks*, and $I2P(v_{25}) =$ *apple* is associated with the category *electronics*.

4 TIKRQ PROCESSING FRAMEWORK

In this section, we propose our Set-Based Search Algorithm to find the resulting routes. Before we present the algorithm, we extend the concept of skeleton distance [29] to skeleton time which will be used in our pruning rules. Given two indoor items x_i and x_j, the skeleton time $\gamma(x_i, x_j)_L$ can be used as a lower bound of the time needed from x_i to x_j.

$$\gamma(x_i, x_j)_L = \begin{cases} \frac{|x_i, x_j|_E}{s_{walk}}, & \text{if } x_i \text{ and } x_j \text{ are on the same floor;} \\ \min\Bigg(\min_{\substack{sd_i \in SD(x_i),\\ sd_j \in SD(x_j)}} \frac{|x_i, sd_i|_E + \delta_{s2s}(sd_i, sd_j) + |sd_j, x_j|_E}{s_{walk}}, \\ \min_{\substack{ed_i \in ED(x_i),\\ ed_j \in ED(x_j)}} \Big(\gamma(x_i, ed_i)_L + \\ \gamma_{e2e}(ed_i, ed_j) + \gamma(ed_j, x_j)_L\Big)\Bigg), & \text{otherwise.} \end{cases}$$

where $\gamma(x_i, x_j)_L$ is the time needed to walk in Euclidean distance from x_i to x_j if they are on the same floor. Otherwise, we find the time needed for the fastest path that goes through the staircase doors (e.g., $sd_i \in SD(x_i)$ and $sd_j \in SD(x_j)$) or the transport doors (e.g., $ed_i \in ED(x_i)$ and $ed_j \in ED(x_j)$) to reach x_j from x_i.

4.1 Set-Based Search Algorithm (SSA)

We give the following observation which provides a clue to developing an efficient algorithm for TIKRQ.

Observation 1 (Partition Set). *Given a set S of key partitions, any route R formed by the partitions in S has the same partition cost and textual relevance.* □

With a slight abuse of notations, we denote the partition cost and textual relevance of a set S of key partitions by $PC(S)$ and $\rho(S)$, respectively. It is easy to see that both metrics are not affected by the *order* of visiting, and thus $PC(S) = PC(R)$ and $\rho(S) = \rho(R)$ for any route R formed by the key partitions in S. Based on this observation, we propose a set-based search algorithm *SSA* as follows.

High Level Idea. This algorithm searches for the resulting routes by focusing on the partition sets, as shown in Figure 3. For each set S of key partitions, we check whether any feasible route R exist (i.e., $\gamma(R) < \Delta_{Max}$). If such a route exists, the top-k results are updated accordingly.

The advantage of *SSA* is that it *separates* the time needed of a route R from its cost part. Thus, effective pruning techniques based on partition cost and textual relevance can be applied to

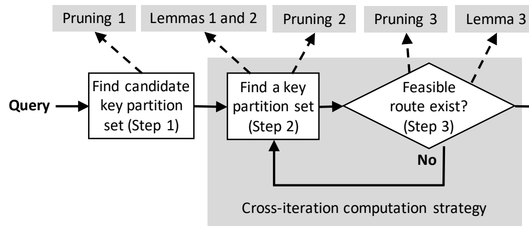

Pruning 1
Lemmas 1 and 2
Pruning 2
Pruning 3
Lemma 3
Query
Find candidate key partition set (Step 1)
Find a key partition set (Step 2)
Feasible route exist? (Step 3)
Yes
No
Maintain top-k results
Cross-iteration computation strategy

Figure 3: Flow of Set-based Search Algorithm

filter out unpromising routes quickly, without performing the time-consuming route search and expansion. Compared to the graph-based algorithms in [9], *SSA* maps *multiple* routes into *one* set, resulting in a much smaller search space. Note that this search strategy naturally conforms with our requirement of the unique partition set, which improves the diversity of our top-k results.

Specifically, *SSA* maintains a list *TopKRoutes* storing the current top-k best feasible routes, and *curKCost* storing the cost of the k-th route found so far. It has four major steps.

- *Step 1 (Candidate Key Partition Set Finding):* Find the candidate key partition set (*CKP*) of candidate key partitions from the set of query keywords QW.
- *Step 2 (Key Partition Set Finding):* Find a set S of key partition set from *CKP* to be the key partition set of a route R to be found.
- *Step 3 (Feasible Route Finding):* Find a feasible route R which starts from p_s, passes all key partitions in S and ends at p_t (if any), and update *TopKRoutes* with R correspondingly if $cost(R) < curKCost$.
- *Step 4 (Iterative Step):* Resume Step 2 until all key partition sets are traversed.

The above search strategy is based on the set of all possible combinations of *CKP*. A straightforward implementation of this strategy would enumerate $2^{|CKP|}$ key partition sets, and each set would have $|S|!$ possible routes. This is prohibitively expensive in practice. Thus, we need a careful design to prune the search space effectively. In the following, we discuss the pruning techniques enjoyed by *SSA*.

4.1.1 Pruning at Step 1.

Pruning Rule 1 (Candidate Key Partitions). *For a partition v_i in a key partition set S, if its time cost lower bound $LB(\gamma(v_i)) > \Delta_{Max}$, then v_i can be pruned, where*

$$LB(\gamma(v_i)) = \gamma(p_s, v_i, p_t) + t_{wait}(v_i)$$

$$\gamma(p_s, v_i, p_t) = \min_{\substack{d_i \in P2D_{\sqsupset}(v_i),\\ d_j \in P2D_{\sqsubset}(v_i)}} \left(\gamma(p_s, d_i)_L + \frac{\delta_{d2d}(d_i, d_j)}{s_{walk}} + \gamma(d_j, p_t)_L\right)$$

4.1.2 Pruning at Step 2.

Firstly, we utilize an inverted file indexed by QW to organize *CKP* to avoid generating sets that contain 'unnecessary' key partitions. That is, only the partition sets with each partition *contributing* to a query keyword will be considered. Note that in this way, we also bound the size of each set S to $|QW|$.

Secondly, given a subset S' of the key partition set S to be generated, we impose a cost lower bound $cost_{LB}(S|S')$ of S, as follows.

$$cost_{LB}(S|S') = \alpha \frac{PC(S')}{PC_{max} \cdot |QW|} + (1-\alpha)(1 - \frac{\rho(S') + (|QW| - |S'|)}{|QW|})$$

Lemma 1 (Set Cost). *Let S be a key partition set and $S' \subset S$, we have $cost(S) \geq cost_{LB}(S|S')$.* □

Proof 1. *Since $|S| = |QW| > |S'|$ and each key partition has its relevance $rs \leq 1$ for each query keyword, we have $\rho(S) \leq \rho(S') + (|QW| - |S'|)$. It is easy to see that $PC(S') \leq PC(S)$. Thus, we have $cost(S) > cost_{LB}(S|S')$.* □

The above Lemma suggests that if $cost_{LB}(S|S') > curKCost$, we can terminate the enumeration on S'.

Thirdly, we sort the partitions v_i in each inverted list in ascending order of $f(v_i)$, where

$$f(v_i) = \alpha \frac{PC(v_i)}{PC_{max}} - (1-\alpha)\, v_i.rs$$

Lemma 2 (List Ordering). *Let v_i and v_j be two key partitions in an inverted list with $f(v_i) \leq f(v_j)$, S' be a key partition set containing v_i and $S'' = S' \setminus \{v_i\} \cup \{v_j\}$. Then, we have $cost_{LB}(S|S') \leq cost_{LB}(S|S'')$.* □

Proof 2. *Consider the set $S'_o = S' \setminus \{v_i\}$. It can be proven that $cost_{LB}(S|S'_o) \geq cost_{LB}(S|S') - \frac{f(v_i)}{|QW|}$. Since $S'' = S'_o \cup \{v_j\}$, we have $cost_{LB}(S|S'_o) + \frac{f(v_j)}{|QW|} \geq cost_{LB}(S|S'')$. As $f(v_i) \leq f(v_j)$, we have $cost_{LB}(S|S') \leq cost_{LB}(S|S'')$.* □

By ordering the inverted lists in this way, we can impose an early stopping condition: If $cost_{LB}(S|S') > curKCost$, we can terminate the enumeration of the remaining partitions in the list.

Fourthly, some key partition sets can be excluded by considering their upper bound of total waiting time.

Pruning Rule 2 (Set Waiting Time). *Given a key partition set S, we upper bound S's total waiting time of partitions in S, $\sum_{v \in S} v.waitTime$, by $waitTime_{max}$, which is defined as follows.*

$$waitTime_{max} = \Delta_{Max} - \gamma(p_s, p_t)_L$$

Note that $waitTime_{max}$ can be pre-computed because $\gamma(p_s, p_t)_L$ is identical for all queries with the same pair of p_s and p_t.

4.1.3 Algorithm SSA.

We design *SSA* as shown in Algorithm 1. Specifically, it maintains a list *TopKRoutes* storing the k best-known routes found so far (line 2). Then, it finds the set of candidate key partitions *CKP* (line 4), by utilizing Pruning 1. Next, it performs an iterative process as follows (lines 5 to 11). It iterates through each key partition set S, and if S passes our lower bound cost checking (Lemma 1) and waiting time checking (Pruning 2), it finds the feasible route R of S by $findFeasibleRoute()$ (to be detailed in Algorithm 2). If such a route R exists, we update the *TopKRoutes* by R. The algorithm terminates when all sets have been processed. The *TopKRoutes* is then returned as the result.

One remaining issue is that given a set S, how to efficiently find the feasible route, if it exists. We present an algorithm for that in the following section.

Algorithm 1 SSA $(p_s, p_t, QW, \Delta_{Max}, k)$

```
1:  if $\delta(p_s,p_t) > \Delta_{Max}$ then return $\emptyset$
2:  $TopKRoutes \leftarrow \emptyset$
3:  $waitTime_{max} \leftarrow \Delta_{Max} - \frac{\delta(p_s,p_t)}{s_{walk}}$
4:  $CKP \leftarrow \cup_{w_Q \in QW} \mathcal{CP}(w_Q)$                ▷ Step 1
5:  for each possible subset $S$ of $CKP$ do                      ▷ Step 2
6:      if $cost_{LB}(S) > curKCost$ then continue;
7:      if $\sum_{v \in S} v.waitTime > waitTime_{max}$ then continue;
8:      $R \leftarrow findFeasibleRoute(S, p_s, p_t, \Delta_{Max})$     ▷ Step 3
9:      if $R \neq \emptyset$ then
10:         update $TopKRoutes$ with $R$
11:         $curKCost \leftarrow$ cost of the $k$-th route in $TopKRoutes$
12: return $TopKRoutes$
```

4.2 Feasible Route Search

Given a set S of partitions, we want to find a feasible route from p_s to p_t that passes all partitions in S. A naive approach is to try all permutations of the partitions in S. Instead, we propose a best-first search algorithm that can find the feasible route efficiently. It returns the feasible route R if it exists. Otherwise, it returns $\emptyset$.

Before we present the algorithm, we introduce some pruning techniques to speed up the feasible route search. First, not all partial routes need to be explored. In particular, given a partial route $R = \{p_s, ..., d_k\}$, we compute its lower bound time cost and introduce a pruning as follows.

Pruning Rule 3 (Route Time Cost). *A partial route $R = \{p_s, ..., d_k\}$ can be pruned if $\gamma_{LB}(R) = \gamma(R) + \gamma(d_k, p_t)_L \geq \Delta_{Max}$.* □

Second, not all doors in each key partition need to be considered when expanding a partial route. Given a key partition v', a partial route $R = \{p_s, ..., d_{y-1}, d_y\}$ that has expanded to v', if both d_{y-1} and d_y are connected to v', R' can be safely discarded. Formally, we have the following lemma.

Lemma 3 (Route Pruning). *Consider a partial route $R' = \{p_s, .., d_{y-1}, d_y\}$, where $d_{y-1}, d_y \in P2D_{\sqsupset}(v')$. There exists a route R that connects to v' and is faster than R'.* □

Proof 3. *Consider another partial route $R = \{p_s, .., d_{y-1}\}$. It is easy to see that $\gamma(R') \geq \gamma(R)$ since $\gamma(R') = \gamma(R) + \frac{|d_{y-1}, d_y|_E}{s_{walk}}$.* □

Based on the above Lemma, a partial route $R' = \{p_s, .., d_{y-1}, d_y\}$ can be pruned if $d_{y-1}, d_y \in P2D_{\sqsupset}(v')$.

Algorithm 2 presents the $findFeasibleRoute$ algorithm. Specifically, a minimum priority queue Q (initialized in line 1) is used to handle the order of route expansion. An element in Q is a four-tuple (v, R, γ_{LB}, S') that stores the local information of the current partial route, where v is the last partition that R reaches, $R = \{p_s, d_i, ..., d_k\}$ is the partial route that has been expanded so far, γ_{LB} is the lower bound time cost of $R \cup \{p_t\}$, and S' is the set of remaining partitions that have not been explored by R yet. The elements in Q are sorted in ascending order of $|S'|$.

The algorithm initializes a route R_0 by p_s and puts it into Q (lines 2 to 3). It then performs the expansion iteratively (lines 4 to 18). In each iteration, it pops out the element (v, R, γ_{LB}, S') with the smallest $|S'|$ from Q (line 5), and check if S' is empty. If so,

Algorithm 2 findFeasibleRoute $(S = (v_1, v_2, ..., v_n), p_s, p_t, \Delta_{Max})$

```
1:  Initialize a priority queue $Q$
2:  $R_0 \leftarrow (p_s)$
3:  $Q.push(v(p_s), R_0, 0, S)$
4:  while $Q$ is not empty do
5:      $(v, R, \gamma_{LB}, S') \leftarrow Q.pop()$
6:      $d_k = R.tail$
7:      if $S' = \emptyset$ then
8:          find fastest route from $d_k$ to $p_t$
9:          $R' \leftarrow$ append $(d_k, ..., p_t)$ to $R$
10:         if $\gamma(R') \leq \Delta_{Max}$ then return $R'$
11:     for each $v' \in S'$ do
12:         for each $d_y \in P2D_{\sqsupset}(v')$ do
13:             find fastest route from $d_k$ to $d_y$
14:             if $d_{y-1} \in P2D_{\sqsupset}(v')$ then continue;
15:             $R' \leftarrow$ append $(d_k, ..., d_y)$ to $R$
16:             $\gamma'_{LB} \leftarrow \gamma(R') + \gamma(d_y, p_t)_L$
17:             if $\gamma'_{LB} \leq \Delta_{Max}$ then
18:                 $Q.push(v', R', \gamma'_{LB}, S' \setminus \{v'\})$
19: return $\emptyset$
```

all key partitions in (the original) S is covered by R, and it connects R from d_k to p_t to form a complete route R'. If $\gamma(R') < \Delta_{Max}$, it returns R' immediately as R' is a feasible route (lines 7 to 10). Otherwise, it expands the current partial route to cover a key partition $v' \in S'$ (lines 11 to 18). For each $d_y \in P2D_{\sqsupset}(v')$, it finds the fastest route from d_k to d_y, checks if the route passes the checking (Lemma 3). If so, it generates a new route R' by appending the fastest route to R, and pushes it to Q if it passes the time constraint checking (Pruning 3). The expansion continues until all elements in Q have been processed. It returns $\emptyset$ if no feasible route can be found.

Cross-Iteration Computation Strategy. To further speed up the algorithm, we have the following strategy to store and reuse the information computed in the current iteration for future iterations.

Consider an iteration. Given two doors d_i and d_j that we have processed, we maintain the fastest partial route information from d_i to d_j in a global hashmap H_{fpr}, to avoid re-computation in the future iterations. In particular, when we find the fastest route from d_i and d_j (line 13), we check whether $key = (d_i, d_j)$ exists in H_{fpr}. If so, we can append the saved route to the current route directly. Otherwise, we proceed to search for the route. Once such a route is found, it is inserted into H_{fpr}.

Moreover, we maintain another global hashmap H_{in} to store those (partial) routes that are found to be infeasible. To illustrate, consider an example with $S = \{v_1, v_2, v_3\}$. If we found that there does not exist a feasible (partial) route R that contains $\langle p_s, v_1, v_2\rangle$ when we process S, we add $key = \langle v_1, v_2\rangle$ into H_{in}. Then, when we find the feasible route for another set $S' = \{v_1, v_2, v_4\}$, we do not need to consider R that contains $\langle p_s, v_1, v_2\rangle$ since it must also be infeasible. Formally, we perform a checking when we search for the feasible route for a set S as follows. If any route R_{in} is in H_{in}, we know that R_{in} is infeasible and return $\emptyset$ immediately. Note that H_{in} can be updated accordingly when we check the feasibility of new partial routes to a remaining key partition. If no new partial route satisfies the time constraint checking, R_{in} is inserted into H_{in}.

4.3 Time Complexity

The time complexity of *SSA* is dominated by the key partition sets processing part (lines 5 to 18 in Algorithm 1). Let $|KPS|$ be the number of key partition sets processed in *SSA* and θ be the time complexity of executing one iteration. The time complexity of *SSA* is $O(|KPS| \cdot \theta)$. In practice, we have $|KPS| << 2^{|CKP|}$ since it utilizes the pruning techniques.

Consider θ. It is dominated by the time cost of executing the $findFeasibleRoute()$ method (i.e., Algorithm 2). Let $|d_{max}|$ be the maximum number of doors a partition has, and m be the cost for computing the fastest route from a door to another. The time cost is $O(|S|! \cdot |S| \cdot |d_{max}|^2 \cdot m)$, where $|S| = |QW|$, since the number of possible permutations of S is $O(|S|!)$, and the time complexity of computing a complete route for one permutation is $O(|S| \cdot |d_{max}|^2 \cdot m)$. Note that in practice the running time should be much faster since our cross-iteration computation strategy can help to reduce the computation needed. In summary, the time complexity of *SSA* is $O(|KPS| \cdot |S|! \cdot |S| \cdot |d_{max}|^2 \cdot m)$.

5 EMPIRICAL STUDIES

5.1 Experimental Set-up

Indoor Space. Following [9], we generate a n-floor building based on a real world floor plan [5], where $n = \{3, 5, 7, 9\}$. Each floor is 1368m × 1368m, consists of 96 rooms, 4 hallways, and 4 staircases. We obtain 141 partitions and 200 doors on each floor by decomposing those irregular hallways into smaller and regular partitions. To model the *elevators* in an indoor space, we convert two staircases to elevators that connect to all floors, rather than the adjacent floors only. Each elevator has a waiting time of 30 seconds and takes 10 seconds to traverse from one floor to another. The remaining two staircases connect two adjacent floors, each being 20m long. By default, we set $n = 5$ and the indoor space contains 705 partitions and 1100 doors.

Partition Keywords. We assign keywords to each room as follows. First, we crawl the shop information of five shopping malls in Hong Kong [6] online using Scrapy. We obtain 2074 documents for 1225 shop brands. All the 1225 brand names are used as i-words. Second, we manually categorize these brand names into 11 categories, following the categorization used in the shopping malls (e.g., *clothing*, *cosmetics* and *restaurant*). These categories are used as the c-words. Each category contains 111 i-words on average. Third, we feed these i-words into the RAKE algorithm [22] to extract keywords from the corresponding 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. In total, we have 9195 extracted keywords and each i-word corresponds to an average of 16.6 extracted keywords. For the partitions, their static cost and waiting time are picked uniformly at random, in the range $[1, 10]$ and $[0, 100]$, respectively.

[5]https://longaspire.github.io/s/fp.html
[6]https://longaspire.github.io/s/hkdata.html

Queries. We generate the query keywords as follows. The number of query keywords $|QW|$ is in the range $[1, 5]$, as over 95% of web search queries have at most 5 keywords [7]. We vary the fractions of c-words/i-words/t-words in QW, as the parameter $c/i/t$. The procedure is to vary the fraction of one type with the other two types being changed accordingly. Take the c-word as an example, we vary its fraction from $p = [0.1, 0.9]$, and the fractions of i-words and t-words are both set to be $(1 - p)/2$. In addition, we vary the parameters α in Equation 1. Table 3 summaries the parameters setting with default values in bold.

Table 3: Parameter Settings

Parameters	Settings
k	1, 3, 5, **7**, 9, 11
$\|QW\|$	1, 2, 3, **4**, 5
Δ_{Max}	3000, **3500**, 4000, 4500, 5000 (seconds)
$c/i/t$	0.1/0.45/0.45, **0.2/0.4/0.4**, . . ., 0.05/0.05/0.9
n	3, **5**, 7, 9
α	0.1, 0.3, **0.5**, 0.7, 0.9

Algorithms. We compare our *SSA* algorithm with a baseline algorithm *SSA\P*. *SSA\P* follows the workflow of *SSA*, but the proposed pruning features and computation strategy in *SSA* are removed. Also, we adapt the algorithm *KoE* [9], which is originally designed for IKRQ. The adaption is as follows. It expands the partial routes from p_s to search one of the key partitions that can cover some of those uncovered query keywords, until all keywords are covered, and finally connects to p_t. For each complete route, it calculates the cost of the routes and maintains the top-k feasible routes. The route-based speed-up techniques (i.e., Pruning 3 and Lemma 3) and cross-iteration computation strategy are also employed in the adaption to allow a fair comparison. All algorithms are implemented in Java and run on a Mac with a 2GHz Quad-Core Intel i5 CPU and 16GB memory.

Performance Metrics. We measure the running time and the memory cost. For each experimental setting, we generate 10 queries and report the average results. Note that the results are based on the queries with routes returned only. In case of no route is returned as the result, we simply re-generate a new query.

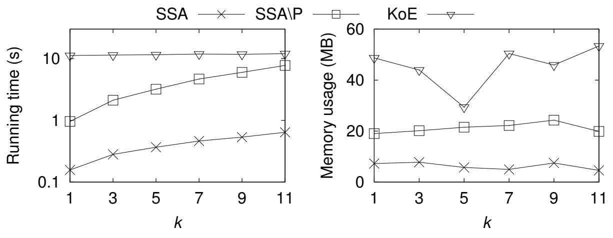
5.2 Experimental Result

5.2.1 *Efficiency Studies.*

Effect of k. Figure 4 shows the results of varying k. According to Figure 4(a), the running time of *SSA* and *SSA\P* increases slightly when k increases. This is because a larger k incurs more routes to be explored. *KoE* is insensitive to k while it always requires significantly long time to terminate. In general, our *SSA* runs faster than *SSA\P* and *KoE* by an order of magnitude, as contributed by the pruning techniques and computation strategy employed. According to Figure 4(b), the memory usage of all algorithms fluctuates with a varying k. *SSA* still consumes less than 10MB of memory in different k values, much less than its competitors.

Effect of Query Size $|QW|$. Figure 5 shows the results of varying the number of query keywords from 1 to 5. According to Figure 5(a), the running time of all algorithms increases with an increasing

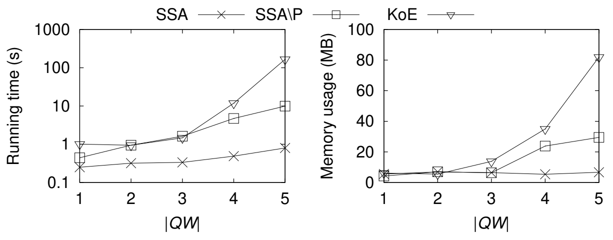
[7]http://www.keyworddiscovery.com/keyword-stats.html



(a) Running time (b) Memory usage

Figure 4: Effect of k

$|QW|$. A larger $|QW|$ leads to more relevant partitions and thus more key partition sets need to be formed and considered. Moreover, each key partition set would be larger and therefore it takes more time to find the complete route for the set in each iteration. Our *SSA* runs consistently faster than *SSA\P* and *KoE*, and the gap enlarges when $|QW|$ increases. This is because our pruning strategies are more effective when $|QW|$ is larger. According to Figure 5(b), the memory usages of all algorithms are similar and increase steadily with $|QW|$. However, *SSA* and *SSA\P* grow slower than *KoE*. Even with more pruning strategies employed, *SSA* consumes fewer memories than *SSA\P* and *KoE* for $|QW|$. This is due to the use of the cross-iteration computation strategy.

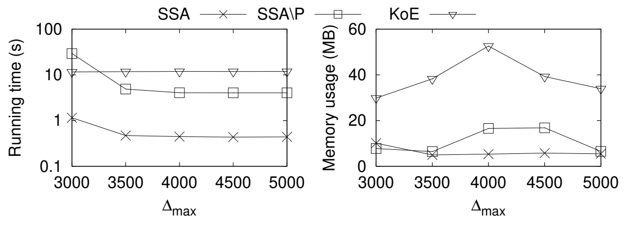


(a) Running time (b) Memory usage

Figure 5: Effect of $|QW|$

Effect of Time Constraint Δ_{Max}. Figure 6 reports the results of varying Δ_{Max}. According to Figure 6(a), the running time of both *SSA* and *SSA\P* decreases when Δ_{Max} increases and our *SSA* always outperforms *SSA\P* and *KoE*. Note that a looser Δ_{Max} reduces the difficulty of finding the feasible routes and results in fewer key partition sets to explore. In this sense, the effectiveness of Pruning Rules 2 and 3 is amplified in a setting of a larger Δ_{Max}. Referring to Figure 6(b), the memory usage of *SSA* decreases when Δ_{Max} increases, since fewer paths and infeasible sets need to be stored when a larger Δ_{Max} is set. On the other hand, both *SSA\P* and *KoE* are insensitive to Δ_{Max}, and incur higher memory usages than *SSA*.

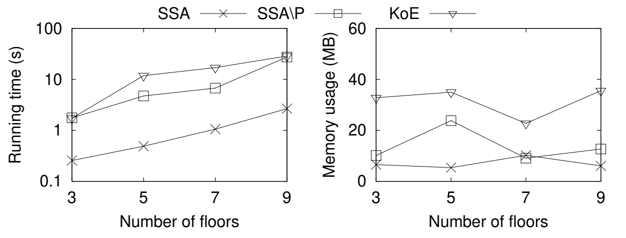
Effect of Number of Floors n. To evaluate the scalability of our algorithm, we vary n in $\{3, 5, 7, 9\}$ and report the result in Figure 7. According to Figure 7(a), the running time of all algorithms increases with n increases. A higher n means a larger number of partitions and thus more relevant partitions need to be checked. Particularly, the elevators in our setting allow the route to pass



(a) Running time (b) Memory usage

Figure 6: Effect of Δ_{Max}

different floors easily. Thus, the time constraint can barely help reducing the search space. Nevertheless, *SSA* runs consistently faster than its competitors and it can finish within 3 seconds when n grows up to 9, showing its scalability.



(a) Running time (b) Memory usage

Figure 7: Effect of n

Effect of α. Figure 8 reports the running time of the algorithms when varying α. Our *SSA* runs faster than *SSA\P* and *KoE* by orders of magnitude. Also, the running time of *KoE* decreases when α increases. For the route cost given in Definition 1, having a larger α puts less weight on routes' textual relevance, which is easier for a graph-based algorithm like *KoE* to find routes with smaller cost. The memory usages of all algorithms are insensitive to α, and *SSA* uses the least memory among all three. We omit the figure here due to the page limit.

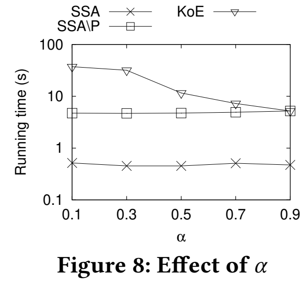


Figure 8: Effect of α

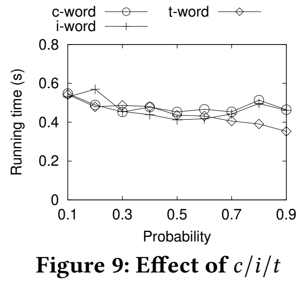


Figure 9: Effect of $c/i/t$

Effect of Fraction of c/i/t-words. We vary the parameter p of each type of word (cf. Section 5.1) and report the result for *SSA* in Figure 9. In general, there is not much difference in the running time of different combinations of keywords.

5.2.2 *Effectiveness Studies.*

Case Study. To show that TIKRQ is able to return desirable routes in practice, we perform a case study by comparing the TIKRQ result with that of minimizing the route's time cost. The query keywords are *apple* (an i-word) and *coffee* (a t-word), $\Delta_{Max} = 3600$ (one hour), $k = 3$. We use $\alpha = 0.8$ to reflect and suit the needs of keyword-awareness in shopping. The routes returned by TIKRQ are listed in Table 4. The top-1 route (say R) has a total route cost $Cost(R) = 0.125$ and time cost $\gamma(R) = 2474.545$ ($\approx$ 41 minutes). In particular, R has a textual relevance of 1, as it passes the partitions *apple* and *coffee day* (which sells coffee). In contrast, the route R' with the minimum time cost (i.e., $\gamma(R') = 1596.807 \approx 27$ minutes) incurs an overall route cost of 0.450. R' only has a textual relevance of 0.55, as it does not pass *apple* but another partition with category *electronics*. Although R has a longer time needed, its textual relevance and route cost meet the practical user needs. This demonstrates that our returned paths can better serve the users in the context of keyword awareness.

Table 4: Case Study

	k	Textual Relevance	Route Cost	Time Cost
SSA returned path	1	1	0.05	2474.545
	2	1	0.08	1891.023
	3	1	0.13	2540.905
minimum time cost path	R'	0.55	0.45	1596.807

Effect of Unique Partition Set. To compare the results with and without adopting the concept unique partition set, we run $SSA\backslash UPS$ which removed the requirement of unique partition set. In other words, it allows different routes in the k resulting routes to have an identical key partition set.

We measure the identical rate as the fraction of routes with the identical key partition set with others. We ran 10 queries for each k and Figure 10 reports the average rate. It shows that the identical rate of $SSA\backslash UPS$ increases rapidly when k increases. More than 60% of the returned routes have identical key partition sets when $k \geq 5$. Such routes are not interesting to users and hinder the diversity of the results. This verifies that the unique partition set offers users more diversified combinations of partitions in the result.

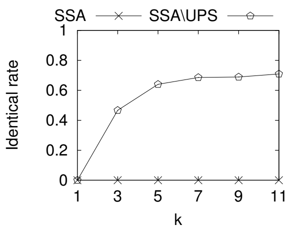


Figure 10: Identical Rate

6 RELATED WORK

Query Processing in Indoor Space. Efficient indoor query processing has received significant attention in recent years. Some works [18, 30, 32, 33] studied the indoor spatial queries such as range queries, kNN queries, and shortest path queries under various settings. Lu et al. [17] designed an indoor space model that facilitates indoor shortest path finding. Shao et al. [26, 27] proposed the VIP-Tree and KP-Tree that allows efficient processing of indoor shortest path queries and spatial keyword queries. However, they did not consider the keyword-aware routing queries, which is the focus of this work. Luo et al. [19] studied the time-constrained sequence route query in an indoor space. Their work considers the stay-time and partitions' category, but not the objects' static cost and thus their solution is not applicable to our problem. Liu et al. [16] studied the indoor temporal-aware shortest path query, which considers the current time stamp and the opening hours of the doors. The temporal-aware setting is orthogonal to our problem, and it can be integrated into our problem to model the case that partitions and doors have different opening hours.

Keyword-aware Routing in Indoor Space. Salgado [23] studied the keyword-aware skyline route (KSR) search in indoor venues that considers the number of objects in the routes and the route distances. While KSR assumes that each partition contains one keyword, our setting allows a partition to have multiple keywords. Salgado et al. [24] studied the category-aware multi-criteria indoor route planning queries (CAM queries) that consider the objects' keyword and static cost. Shao et al. [25] studied the indoor trip planning queries (iTPQ) and developed a solution called VIP-tree neighbor expansion that exploits the features of indoor space, such as room and hallways. Recently, Feng et al. [9] studied the IKRQ problem, which finds k s-to-t routes with the highest scores that consider both the keyword relevance and spatial distance. Each route has a distance satisfying a pre-defined distance constraint. They defined prime route to return diverse results, and developed two algorithms for answering the query. All of these works fail to capture different criteria at the same time and thus their solutions can not be directly applied to our problem. Table 5 shows the summary and comparison of all those proposals and our TIKRQ.

Table 5: Existing Indoor Keyword-aware Routing Queries

	Textual Constraint	Spatial Constraint	Static Cost	Result Diversification
KSR [23]	boolean	distance	-	Skyline
CAM [24]	boolean	distance	√	-
iTPQ [25]	boolean	distance	-	-
IKRQ [9]	i/t-word	distance	-	Prime route
TIKRQ	i/t/c-word	time budget	√	Unique partition set

Outdoor Keyword-aware Query and Routing. Some works [3, 6, 7, 21] studied the outdoor spatial keyword queries that retrieve a single object close to the query location and relevant to the query keywords, while others [4, 10, 31] find an object set as a solution.

Given a source point s, a target point t, and a category set C, the trip planning query [11] finds the shortest s-to-t route that passes at least one object from each category in C. The optimal sequenced route query [28] finds the shortest route that passes the categories in the user-specified sequence, while others [5, 15] consider partial order. The keyword-aware optimal route query [2] finds a route that covers all query keywords, has the minimum objective score, and meets the given budget constraint. The optimal route search [34] finds a route that has maximum query keywords coverage and satisfies the budget constraint. The clue-based route search [36] allows users to specify the order of keywords to cover and the distance range from one matched keyword to the next one. All these works fall short for indoor topology considered in our TIKRQ problem. Also, none of them organize the keywords according to their semantics.

7 CONCLUSION AND FUTURE WORK

In this paper, we studied the problem of time-constrained indoor keyword-aware routing, which find routes with minimum route costs while satisfying a time constraint. In our setting, the route costs capture both static cost and textual relevance of the route to meet the practical user needs. We developed a set-based search algorithm *SSA* to answer the query. Extensive experiments were conducted that verified the efficiency of *SSA*.

For future work, we can take the opening hours of partitions and doors into account. It is also interesting to study the hierarchical word organization for partitions and to provide suggestions to refine query keywords when no route is found. Moreover, it is relevant to consider additional constraints like prohibiting staircases in a route and allowing the user to specify a preferred visiting order.

ACKNOWLEDGMENT

This work was supported by Independent Research Fund Denmark (No. 8022-00366B).

REFERENCES

[1] Anahid Basiri, Elena Simona Lohan, Terry Moore, Adam Winstanley, Pekka Peltola, Chris Hill, Pouria Amirian, and Pedro Figueiredo e Silva. 2017. Indoor location based services challenges, requirements and usability of current solutions. *Computer Science Review* 24 (2017), 1–12.

[2] Xin Cao, Lisi Chen, Gao Cong, and Xiaokui Xiao. 2012. Keyword-aware optimal route search. *PVLDB* 5, 11 (2012), 1136–1147.

[3] Ariel Cary, Ouri Wolfson, and Naphtali Rishe. 2010. Efficient and scalable method for processing top-k spatial boolean queries. In *SSDBM*. Springer, 87–95.

[4] Harry Kai-Ho Chan, Cheng Long, and Raymond Chi-Wing Wong. 2018. On generalizing collective spatial keyword queries. *TKDE* 30, 9 (2018), 1712–1726.

[5] Haiquan Chen, Wei-Shinn Ku, Min-Te Sun, and Roger Zimmermann. 2008. The multi-rule partial sequenced route query. In *SIGSPATIAL*. ACM, 10.

[6] Gao Cong, Christian S Jensen, and Dingming Wu. 2009. Efficient retrieval of the top-k most relevant spatial web objects. *PVLDB* 2, 1 (2009), 337–348.

[7] Ian De Felipe, Vagelis Hristidis, and Naphtali Rishe. 2008. Keyword search on spatial databases. In *ICDE*. IEEE, 656–665.

[8] Georgios J Fakas, Yilun Cai, Zhi Cai, and Nikos Mamoulis. 2018. Thematic ranking of object summaries for keyword search. *DKE* 113 (2018), 1–17.

[9] Zijin Feng, Tiantian Liu, Huan Li, Hua Lu, Lidan Shou, and Jianliang Xu. 2020. Indoor Top-k Keyword-aware Routing Query. In *ICDE*. IEEE, 1213–1224.

[10] Tao Guo, Xin Cao, and Gao Cong. 2015. Efficient Algorithms for Answering the m-Closest Keywords Query. In *SIGMOD*. ACM, 405–418.

[11] Feifei Li, Dihan Cheng, Marios Hadjieleftheriou, George Kollios, and Shang-Hua Teng. 2005. On trip planning queries in spatial databases. In *SSTD*. Springer, 273–290.

[12] Huan Li, Hua Lu, Muhammad Aamir Cheema, Lidan Shou, and Gang Chen. 2020. Indoor Mobility Semantics Annotation Using Coupled Conditional Markov Networks. In *ICDE*. IEEE, 1441–1452.

[13] Huan Li, Hua Lu, Lidan Shou, Gang Chen, and Ke Chen. 2018. Finding most popular indoor semantic locations using uncertain mobility data. *TKDE* 31, 11 (2018), 2108–2123.

[14] Huan Li, Hua Lu, Lidan Shou, Gang Chen, and Ke Chen. 2018. In search of indoor dense regions: An approach using indoor positioning data. *TKDE* 30, 8 (2018), 1481–1495.

[15] Jing Li, Yin David Yang, and Nikos Mamoulis. 2012. Optimal route queries with arbitrary order constraints. *TKDE* 25, 5 (2012), 1097–1110.

[16] Tiantian Liu, Zijin Feng, Huan Li, Hua Lu, Muhammad Aamir Cheema, Hong Cheng, and Jianliang Xu. 2020. Shortest Path Queries for Indoor Venues with Temporal Variations. In *ICDE*. IEEE, 2014–2017.

[17] Hua Lu, Xin Cao, and Christian S Jensen. 2012. A foundation for efficient indoor distance-aware query processing. In *ICDE*. IEEE, 438–449.

[18] Hua Lu, Bin Yang, and Christian S Jensen. 2011. Spatio-temporal joins on symbolic indoor tracking data. In *ICDE*. IEEE, 816–827.

[19] Wenyi Luo, Peiquan Jin, and Lihua Yue. 2016. Time-Constrained Sequenced Route Query in Indoor Spaces. In *APWeb*. Springer, 129–140.

[20] Lu Qin, Jeffrey Xu Yu, and Lijun Chang. 2012. Diversifying top-k results. *PVLDB* 5, 11 (2012), 1124–1135.

[21] Joao B Rocha-Junior, Orestis Gkorgkas, Simon Jonassen, and Kjetil Nørvåg. 2011. Efficient processing of top-k spatial keyword queries. In *SSTD*. Springer, 205–222.

[22] Stuart Rose, Dave Engel, Nick Cramer, and Wendy Cowley. 2010. Automatic keyword extraction from individual documents. *Text mining: applications and theory* 1 (2010), 1–20.

[23] Chaluka Salgado. 2018. Keyword-aware skyline routes search in indoor venues. In *SIGSPATIAL-ISA*. 25–31.

[24] Chaluka Salgado, Muhammad Aamir Cheema, and David Taniar. 2018. An efficient approximation algorithm for multi-criteria indoor route planning queries. In *SIGSPATIAL*. 448–451.

[25] Zhou Shao, Muhammad Aamir Cheema, and David Taniar. 2018. Trip planning queries in indoor venues. *Comput. J.* 61, 3 (2018), 409–426.

[26] Zhou Shao, Muhammad Aamir Cheema, David Taniar, and Hua Lu. 2016. VIP-tree: an effective index for indoor spatial queries. *PVLDB* 10, 4 (2016), 325–336.

[27] Zhou Shao, Muhammad Aamir Cheema, David Taniar, Hua Lu, and Shiyu Yang. 2020. Efficiently Processing Spatial and Keyword Queries in Indoor Venues. *TKDE* (2020).

[28] Mehdi Sharifzadeh, Mohammad Kolahdouzan, and Cyrus Shahabi. 2008. The optimal sequenced route query. *VLDBJ* 17, 4 (2008), 765–787.

[29] Xike Xie, Hua Lu, and Torben Bach Pedersen. 2013. Efficient distance-aware query evaluation on indoor moving objects. In *ICDE*. IEEE, 434–445.

[30] Xike Xie, Hua Lu, and Torben Bach Pedersen. 2014. Distance-aware join for indoor moving objects. *TKDE* 27, 2 (2014), 428–442.

[31] Hongfei Xu, Yu Gu, Yu Sun, Jianzhong Qi, Ge Yu, and Rui Zhang. 2020. Efficient processing of moving collective spatial keyword queries. *VLDBJ* 29, 4 (2020), 841–865.

[32] Bin Yang, Hua Lu, and Christian S Jensen. 2009. Scalable continuous range monitoring of moving objects in symbolic indoor space. In *CIKM*. 671–680.

[33] Wenjie Yuan and Markus Schneider. 2010. Supporting continuous range queries in indoor space. In *MDM*. IEEE, 209–214.

[34] Yifeng Zeng, Xuefeng Chen, Xin Cao, Shengchao Qin, Marc Cavazza, and Yanping Xiang. 2015. Optimal Route Search with the Coverage of Users' Preferences.. In *IJCAI*. 2118–2124.

[35] Chengyuan Zhang, Ying Zhang, Wenjie Zhang, Xuemin Lin, Muhammad Aamir Cheema, and Xiaoyang Wang. 2014. Diversified Spatial Keyword Search On Road Networks.. In *EDBT*. 367–378.

[36] Bolong Zheng, Han Su, Wen Hua, Kai Zheng, Xiaofang Zhou, and Guohui Li. 2017. Efficient clue-based route search on road networks. *TKDE* 29, 9 (2017), 1846–1859.